



- [LTPDA Toolbox](#)
 - [Getting Started with the LTPDA Toolbox](#)
 - [What is the LTPDA Toolbox](#)
 - [System Requirements](#)
 - [Setting-up MATLAB](#)
 - [Starting the LTPDA Toolbox](#)
 - [Trouble-shooting](#)
 - [Examples](#)
 - [Introducing LTPDA Objects](#)
 - [Creating LTPDA Objects](#)
 - [Working with LTPDA objects](#)
 - [Analysis Objects](#)
 - [Creating Analysis Objects](#)
 - [Converting existing data into Analysis Objects](#)
 - [Saving Analysis Objects](#)
 - [Plotting Analysis Objects](#)
 - [Parameter Lists](#)
 - [Creating Parameters](#)
 - [Creating lists of Parameters](#)
 - [Spectral Windows](#)
 - [What are LTPDA spectral windows?](#)
 - [Create spectral windows](#)
 - [Visualising spectral windows](#)
 - [Using spectral windows](#)
 - [Simulation/modelling](#)
 - [Generating model noise](#)
 - [Franklin noise-generator](#)
 - [Pole/Zero Modelling](#)
 - [Creating poles and zeros](#)
 - [Building a model](#)
 - [Model helper GUI](#)
 - [Converting models to IIR filters](#)
 - [Signal Pre-processing in LTPDA](#)
 - [Downsampling data](#)
 - [Upsampling data](#)
 - [Resampling data](#)
 - [Interpolating data](#)
 - [Spikes reduction in data](#)
 - [Data gap filling](#)
 - [Signal Processing in LTPDA](#)
 - [Digital Filtering](#)
 - [IIR Filters](#)
 - [FIR Filters](#)
 - [Spectral Estimation](#)
 - [Power spectral density estimates](#)
 - [Cross-spectral density estimates](#)
 - [Cross coherence estimates](#)
 - [Transfer function estimates](#)
 - [Log-scale power spectral density estimates](#)

- [Log-scale cross-spectral density estimates](#)
 - [Log-scale cross coherence density estimates](#)
 - [Log-scale transfer function estimates](#)
- [Fitting Algorithms](#)
 - [Polynomial Fitting](#)
 - [Time domain Fit](#)
- [Graphical User Interfaces in LTPDA](#)
 - [The LTPDA Launch Bay](#)
 - [The LTPDA Analysis GUI](#)
 - [Basics](#)
 - [Main panel](#)
 - [Parameters](#)
 - [Output panel](#)
 - [Import panel](#)
 - [Repository panel](#)
 - [Nested loops](#)
 - [The LTPDA Repository GUI](#)
 - [The pole/zero model helper](#)
 - [The Spectral Window GUI](#)
 - [The constructor helper](#)
 - [The LTPDA object explorer](#)
 - [The quicklook GUI](#)
- [Working with an LTPDA Repository](#)
 - [What is an LTPDA Repository](#)
 - [Connecting to an LTPDA Repository](#)
 - [Submitting LTPDA objects to a repository](#)
 - [Exploring an LTPDA Repository](#)
 - [Retrieving LTPDA objects from a repository](#)
 - [Using the LTPDA Repository GUI](#)
 - [Connecting to a repository](#)
 - [Submitting objects to a repository](#)
 - [Querying the contents of a repository](#)
 - [Retrieving objects and collections from a repository](#)
- [Class descriptions](#)
 - [ao Class](#)
 - [ssm Class](#)
 - [mfir Class](#)
 - [miir Class](#)
 - [pzmodel Class](#)
 - [timespan Class](#)
 - [plist Class](#)
 - [specwin Class](#)
 - [time Class](#)
 - [pz \(pole/zero\) Class](#)
 - [minfo Class](#)
 - [history Class](#)
 - [provenance Class](#)
 - [param Class](#)
 - [cdata Class](#)
 - [fsdata Class](#)
 - [tsdata Class](#)
 - [xydata Class](#)
 - [xyzdata Class](#)
 - [Constructor Examples](#)

[Constructor examples of the AO class](#)

- [Constructor examples of the MFIR class](#)
- [Constructor examples of the MIIR class](#)
- [Constructor examples of the PLIST class](#)
- [Constructor examples of the PZMODEL class](#)
- [Constructor examples of the SPECWIN class](#)
- [Constructor examples of the TIMESPAN class](#)
- [Functions - By Category](#)
- [Functions - Alphabetical List](#)
- [LTPDA Web Site](#)

LTPDA Toolbox

Welcome to the LTPDA Toolbox for MATLAB. This toolbox provides an object-oriented data analysis environment and is designed to carry out the analysis for the LISA PathFinder mission.

©LTP Team

Getting Started with the LTPDA Toolbox

- [What is the LTPDA Toolbox?](#)
- [System Requirements](#)
- [Setting up MATLAB to work with LTPDA](#)
- [Starting the LTPDA Toolbox](#)

Overview of the main functionality of the Toolbox

Supported platforms, MATLAB versions, and required toolboxes.

Installing and editing the LTPDA Startup file.

What is the LTPDA Toolbox

This section covers the following topics:

- [Overview](#)
- [Features of the LTPDA Toolbox](#)
- [Expected Background for Users](#)

Overview

The LTPDA Toolbox is a MATLAB[®] Toolbox designed for the analysis of data from the LISA Technology Package (part of the LISA Pathfinder Mission). The Toolbox implements accountable and reproducible data analysis within MATLAB[®].

With the LTPDA Toolbox you operate with Analysis Objects. An Analysis Object captures more than just the data from a particular analysis: it also captures the full processing history that led to this particular result, as well as full details of by whom and where the analysis was performed.

Features of the LTPDA Toolbox

The LTPDA Toolbox has the following features:

- Create and process multiple Analysis Objects (AOs).
- Save and load AOs from XML files.
- Plot/view the history of the processing in any particular AO.
- Submit and retrieve AOs to/from an LTPDA Repository.
- Powerful and easy to use Signal Processing capabilities.

Note LTPDA Repositories are external to MATLAB and need to be set up independently.

Expected Background for Users

MATLAB

This documentation assumes you have a basic working understanding of MATLAB. You need to know basic MATLAB syntax for writing your own m-files, and you need to have basic familiarity with the SIMULINK interface to use the LTPDA GUI.

What is the LTPDA Toolbox (LTPDA Toolbox)

System Requirements

The LTPDA Toolbox works with the systems and applications described here:

- [Platforms](#)
- [MATLAB and Related Products](#)
- [Databases](#)

Platforms

The LTPDA Toolbox is expected to run on all of the platforms that support MATLAB, but you cannot run MATLAB with the `-nojvm` startup option.

MATLAB and Related Products

The LTPDA Toolbox requires MATLAB. To use the LTPDA GUI you need SIMULINK. In addition, the following MathWorks Toolboxes are required:

- Signal Processing Toolbox
- Database Toolbox (for LTPDA Repository interaction)

The following versions represent the reference platform against which the LTPDA Toolbox is tested.

Component	Version	Comment
MATLAB	7.6 (R2008a)	
Simulink	7.1	Needed for GUI programming
Signal Processing Toolbox	6.9	
Database Toolbox	3.4.1	Needed to interact with an LTPDA repository
Symbolic Math Toolbox	3.2.3	
Optimisation Toolbox	4.0	

©LTP Team

Setting-up MATLAB

Setting up MATLAB to work properly with the LTPDA Toolbox requires a few steps:

- [Add the LTPDA Toolbox to the MATLAB path](#)
- [Edit the LTPDA startup script](#)
- [Automatically start LTPDA Toolbox on starting MATLAB](#)

Add the LTPDA Toolbox to the MATLAB path

After downloading and un-compressing the LTPDA Toolbox, you should add the directory `ltpda_toolbox` to your MATLAB path. To do this:

- File > Set Path... >
- Choose "Add with Subfolders" and browse to the location of `ltpda_toolbox`
- "Save" your new path. MATLAB may require you to save your new `pathdef.m` to a new location in the case that you don't have write access to the default location. For more details read the documentation on "pathdef" (`>> doc pathdef`).

Edit the LTPDA startup script

The LTPDA Toolbox comes with a default `ltpda_startup.m` file. This may need to be edited for your particular system (though most of the defaults should be fine). To edit this file, do `>> edit ltpda_startup` at the MATLAB command prompt. The following table lists the various variables that are set there:

Variable	Values	Description
USE_LTPDA_PRINT_SETTINGS	'Yes' or 'no'	Choose the LTPDA Print settings, or stick with your own (MATLAB) defaults.
USE_LTPDA_PLOT_SETTINGS	'Yes' or 'no'	Choose the LTPDA Plot settings, or stick with your own (MATLAB) defaults.
REPO_GUI_SERVERLIST	{'localhost', '130.75.117.67', '130.75.117.61'}	A list of default LTPDA repositories that will be presented to the user by the LTPDA Repository GUI. Each hostname is one entry in a cell-array.
REPO_GUI_FONTSIZE	10	A default font-size used on the LTPDA Repository GUI.
PASSWD_WIN_FONTSIZE	10	A default font-size used on the password dialog box for connection to LTPDA Repositories.
TIMEZONE	'UTC'	A default time-zone used when dealing with

		LTPDA Time objects. A list of valid time-zones can be retried with the command: >> <code>java.util.TimeZone.getAvailableIDs</code>
TIME_FORMAT_STR	'yyyy-mm-dd HH:MM:SS.FFF'	A default time format string.
VERBOSE_LEVEL	0-11	The level of terminal output from the toolbox. Higher numbers result in increased terminal output.

Automatically start LTPDA Toolbox on starting MATLAB

In order to automatically load the LTPDA Toolbox settings when MATLAB starts up, add the command `ltpda_startup` to your own `startup.m` file. See >> `doc startup` for more details on installing and editing your own `startup.m` file.

 System Requirements	Starting the LTPDA Toolbox 

©LTP Team

Starting the LTPDA Toolbox

In order to access the functionality of the LTPDA Toolbox, it is necessary to run the command `ltpda_startup`, either at the MATLAB command prompt, or by placing the command in your own `startup.m` file.

The main Graphical User Interface can be started with the command `ltpdagui`. The LTPDA Launch Bay can be started with the command `ltpdalauncher`.

Trouble-shooting

A collection of trouble-shooting steps.

1. Java Heap Problem

When loading or saving large XML files, MATLAB sometimes reports problems due to insufficient heap memory for the Java Virtual Machine.

You can increase the heap space for the Java VM in MATLAB 6.0 and higher by creating a `java.opts` file in the `$MATLAB/bin/$ARCH` (or in the current directory when you start MATLAB) containing the following command:

```
-Xmx$MEMSIZE
```

Recommended:

```
-Xmx536870912
```

which is 512Mb of heap memory.

An additional workaround reported in case the above doesn't work: It sometimes happens with MATLAB R2007b on WinXP that after you create the `java.opts` file, MATLAB won't start (it crashes after the splash-screen).

The workaround is to set an environment variable `MATLAB_RESERVE_LO=0`.

This can be set by performing the following steps:

1. Select Start->Settings->Control Panel->System
2. Select the "Advanced" tab
3. On the bottom, center, click on "Environment variables"
4. Click "New" (choose the one under "User variables for Current User")
5. Enter
Variable Name: `MATLAB_RESERVE_LO`
Variable Value: `0`
6. Click OK as many times as needed to close the window

Then edit/create the `java.opts` file as described above. You can also specify the units (for instance `-Xmx512m` or `-Xmx524288k` or `-Xmx536870912` will all give you 512 Mb).

2. LTPDA Directory Name

Problems have been seen on Windows machines if the LTPDA toolbox directory name contains `'.'`. In this case, just rename the base LTPDA directory before adding it to the MATLAB path.

Examples

General

The directory `examples` in the LTPDA Toolbox contains a large number of example scripts that demonstrate the use of the toolbox for scripting.

You can execute all of these tests by running the command `run_tests`

Constructor examples

[Constructor examples of the AO class](#)

[Constructor examples of the MFIR class](#)

[Constructor examples of the MIIR class](#)

[Constructor examples of the PLIST class](#)

[Constructor examples of the PZMODEL class](#)

[Constructor examples of the SPECWIN class](#)

[Constructor examples of the TIMESPAN class](#)

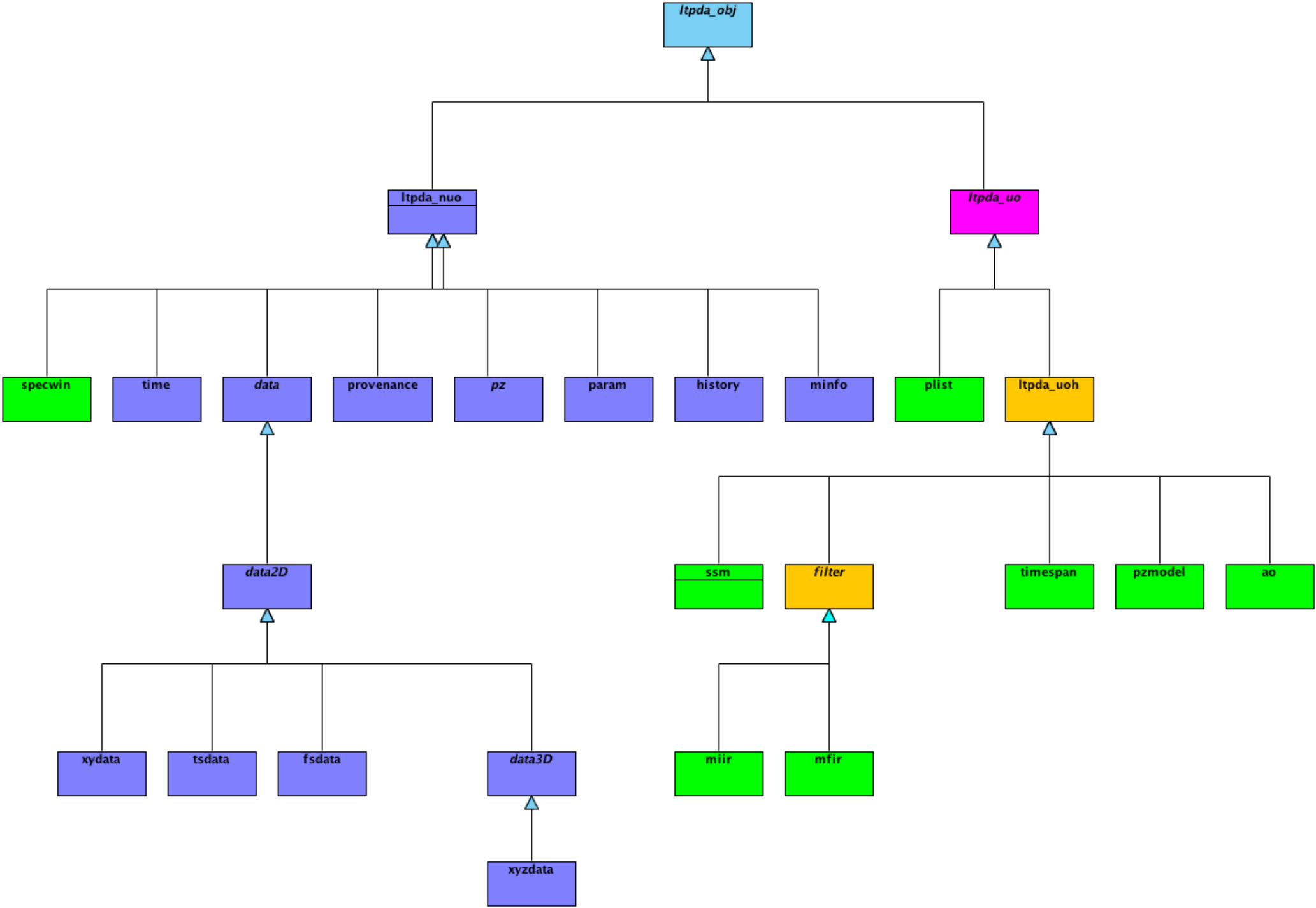
Introducing LTPDA Objects

The LTPDA toolbox is object oriented and as such, extends the MATLAB object types to many others. All data processing is done using objects and methods of those classes.

For full details of objects in MATLAB, refer to [MATLAB Classes and Object-Oriented Programming](#).

LTPDA Classes

Various classes make up the object-oriented infrastructure of LTPDA. The figure below shows all the classes in LTPDA. All classes are derived from the base class, `ltpda_obj`. The classes then fall into two main types deriving from the classes `ltpda_nuo` and `ltpda_uo`.



The left branch, `ltpda_nuo`, are termed 'non-user objects'. These objects are not typically accessed or created by users. The right branch, `ltpda_uo`, are termed 'user objects'. These objects have a 'name' and a 'history' property which means that their processing history is tracked through all LTPDA algorithms. In addition, these 'user objects' can be saved to disk or to an LTPDA repository.

The objects drawn in green are expected to be created by users in scripts or on the LTPDA GUI.

Details of each class are given in:

ao class
History class
Provenance class
tsdata class
fsdata class
xydata class
cdata class
plist class
param class
specwin class
miir class
mfir class
pzmodel class
pole class
zero class
timespan class
time class
timeformat class

Creating LTPDA Objects

Creating LTPDA objects within MATLAB is achieved by calling the constructor of the class of object you want to create. Typically, each class within LTPDA has many possible constructor calls which can produce the objects using different methods and inputs.

For example, if we want to create a parameter list object (`plist`), the help documentation of the `plist` class describes the various constructor methods. Type `help plist` to see the documentation.

Working with LTPDA objects

The use of LTPDA objects requires some understanding of the nature of objects as implemented in MATLAB.

For full details of objects in MATLAB, refer to [MATLAB Classes and Object-Oriented Programming](#). For convenience, the most important aspects in the context of LTPDA are reviewed below.

- [Calling object methods](#)
- [Setting object properties](#)
- [Copying objects](#)

Calling object methods

Each class in LTPDA has a set of methods (functions) which can operate/act on instances of the class (objects). For example, the AO class has a method `psd` which can compute the Power Spectral Density estimate of a time-series AO.

To see which methods a particular class has, use the `methods` command. For example,

```
>> methods('ao')
```

To call a method on an object, `obj.method`, or, `method(obj)`. For example,

```
>> b = a.psd
```

or

```
>> b = psd(a)
```

Additional arguments can be passed to the method (a `plist`, for example), as follows:

```
>> b = a.psd(pl)
```

or

```
>> b = psd(a, pl)
```

In order to pass multiple objects to a method, you must use the form

```
>> b = psd(a1, a2, pl)
```

Some methods can behave as modifiers which means that the object which the method acts on is modified. To modify an object, just give no output. If we start with a time-series AO then modify it with the psd method,

```
>> a
M: running ao/display
----- ao: a -----

name: none
creator: created by hewitson@bobmac.aei.uni-hannover.de[130.75.117.65] on MACI/7.6
(R2008a)/1.9.1 beta (R2008a)
description:
data: (0,-1.75921525387737) (0.1,-0.323940403980841) (0.2,1.70580759558634)
(0.3,0.74566737561773) (0.4,-0.386452719524098) ...
----- tsdata 01 -----

fs: 10
x: [1 100], double
y: [1 100], double
xunits: s
yunits: V
nsecs: 10
t0: 1970-01-01 00:00:00.000
-----

hist: ao / ao / $Id: objects_working_content.html,v 1.3 2008/08/25 21:10:07 hewitson
Exp $-->$Id: objects_working_content.html,v 1.3 2008/08/25 21:10:07 hewitson Exp $
mfilename:
mdlfilename:
-----
```

Then call the psd method:

```
>> a.psd(pl)
M: running ao/psd
M: using default Nfft of 100
M: reset window to BH92(100)
M: using default overlap of 66.1%
M: running ao/display
----- ao: PSD(a) -----

name: PSD(a)
creator: created by hewitson@bobmac.aei.uni-hannover.de[130.75.117.65] on MACI/7.6
(R2008a)/1.9.1 beta (R2008a)
description:
data: (0,0.0488141703757124) (0.1,0.109407517445348) (0.2,0.194309804548859)
(0.3,0.453109075098881) (0.4,0.650807772380848) ...
----- fsdata 01 -----

fs: 10
x: [1 51], double
y: [1 51], double
xunits: empty
yunits: V^2/Hz
t0: 1970-01-01 00:00:00.000
-----

hist: ao / psd / $Id: objects_working_content.html,v 1.3 2008/08/25 21:10:07 hewitson
Exp $
mfilename:
mdlfilename:
-----
```

then the object a is converted to a frequency-series AO.

This modifier behaviour only works with certain methods, in particular, methods requiring more than one input object will not behave as modifiers.

Setting object properties

All object properties must be set using the appropriate setter method. For example, to set the name of a IIR filter object,

```
>> ii = miir();
>> ii.setName('My Filter');
```

Reading the value of a property is achieved by:

```
>> ii.name
ans =
My Filter
```

Copying objects

Since all objects in LTPDA are handle objects, creating copies of objects needs to be done differently than in standard MATLAB. For example,

```
>> a = ao();
>> b = a;
```

in this case, the variable `b` is a copy of the handle `a`, not a copy of the object pointed too by the handle `a`. To see how this behaves,

```
>> a = ao();
>> b = a;
>> b.setName('My Name');
>> a.name

ans =
My Name
```

Copying the object can be achieved using the copy constructor:

```
>> a = ao();
>> b = ao(a);
>> b.setName('My Name');
>> a.name

ans =
none
```

In this case, the variable `b` points to a new distinct copy of the object pointed to by `a`.

©LTP Team

Analysis Objects

Based on the requirement that all results produced by the LTP Data Analysis software must be easily reproducible as well as fully traceable, the idea of implementing analysis objects (AO) as they are described in S2-AEI-TN-3037 arose.

An analysis object contains all information necessary to be able to reproduce a given result. For example

- which raw data was involved (date, channel, time segment, time of retrieval if data can be changed later by new downlinks)
- all operations performed on the data
- the above for all channels of a multi-channel plot

The AO will therefore hold

- the numerical data belonging to the result
- the full processing history needed to reproduce the numerical result

The majority of algorithms in the LTPDA Toolbox will operate on AOs only (these are always methods of the AO class) but there are also utility functions which do not take AOs as inputs, as well as methods of other classes. Functions in the toolbox are designed to be as simple and elementary as possible.

Creating Analysis Objects

Analysis objects can be created in MATLAB in many ways. Apart from being created by the many algorithms in the LTPDA Toolbox, AOs can also be created from initial data or descriptions of data. The various *constructors* are listed in the function help: [ao help](#).

Examples of creating AOs

The following examples show some ways to create Analysis Objects.

- [Creating AOs from text files](#)
- [Creating AOs from XML or MAT files](#)
- [Creating AOs from MATLAB functions](#)
- [Creating AOs from functions of time](#)
- [Creating AOs from window functions](#)
- [Creating AOs from waveform descriptions](#)

Creating AOs from text files.

Analysis Objects can be created from text files containing two columns of ASCII numbers. Files ending in '.txt' or '.dat' will be handled as ASCII file inputs. The first column is taken to be the time instances; the second column is taken to be the amplitude samples. The created AO is of type `tsdata` with the sample rate set by the difference between the time-stamps of the first two samples in the file. The name of the resulting AO is set to the filename (without the file extension). The filename is also stored as a parameter in the history parameter list. The following code shows this in action:

```
>> a = ao('data1.txt')
+ creating AO from text file data1.txt
----- ao: a -----
      tag: -00001
      name: data1
provenance: created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
(R2007a) at 2007-06-23 19:46:05
      comment:
      data: tsdata / data1
      hist: history / ao / $Id: ao.m,v 1.89 2008/03/07 10:02:29 ingo Exp
      mfile:
-----
```

As with most constructor calls, an equivalent action can be achieved using an input [Parameter List](#).

```
>> a = ao(plist('filename', 'data1.txt'))
```

Creating AOs from XML or .mat files

AOs can be saved as both XML and .MAT files. As such, they can also be created from these files.

```
>> a = ao('a.xml')
----- ao: a -----

      tag: -00001
      name: save(data1,a.xml)
(R2007a) provenance: created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
      at 2007-06-23 20:00:21
      comment:
      data: tsdata / data1
      hist: history / ao / $Id: ao.m,v 1.89 2008/03/07 10:02:29 ingo Exp
      mfile:
-----
```

Creating AOs from MATLAB functions

AOs can be created from any valid MATLAB function which returns a vector or matrix of values. For such calls, a parameter list is used as input. For example, the following code creates an AO containing 1000 random numbers:

```
>> a = ao(plist('fcn', 'randn(1000,1)'))
----- ao: a -----

      tag: -00001
      name: randn(1000,1)
(R2007a) provenance: created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
      at 2007-06-23 20:04:52
      comment:
      data: cdata / randn(1000,1)
      hist: history / ao / $Id: ao.m,v 1.89 2008/03/07 10:02:29 ingo Exp
      mfile:
-----
```

Here you can see that the AO is a cdata type and the name is set to be the function that was input.

Creating AOs from functions of time

AOs can be created from any valid MATLAB function which is a function of the variable *t*. For such calls, a parameter list is used as input. For example, the following code creates an AO containing sinusoidal signal at 1Hz with some additional Gaussian noise:

```
pl = plist();
pl = append(pl, 'nsecs', 100);
pl = append(pl, 'fs', 10);
pl = append(pl, 'tsfcn', 'sin(2*pi*1*t)+randn(size(t))');
a = ao(pl)
----- ao: a -----

      tag: -00001
      name: TSfcn
(R2007a) provenance: created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
      at 2007-06-24 06:47:19
      comment:
      data: tsdata / sin(2*pi*1*t)+randn(size(t))
      hist: history / ao / $Id: ao.m,v 1.89 2008/03/07 10:02:29 ingo Exp
      mfile:
-----
```

Here you can see that the AO is a tsdata type, as you would expect. Also note that you need to specify the sample rate (*fs*) and the number of seconds of data you would like to have (*nsecs*).

Creating AOs from window functions

The LTPDA Toolbox contains a class for designing spectral windows (see [Spectral Windows](#)). A spectral window object can also be used to create an Analysis Object as follows:

```
>> w = specwin('Hanning', 1000)
----- Hanning -----

created:
  alpha: 0
  psll: 31.5
  rov: 50
  nenbw: 1.5
  w3db: 1.4382
flatness: -1.4236
  ws: 500
  ws2: 375
  win: 1000

-----

>> a = ao(w)
----- ao: a -----

      tag: -00001
      name: Hanning
provenance: created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
(R2007a) at 2007-06-24 10:27:18
  comment:
    data: cdata / Hanning
    hist: history / ao / $Id: ao.m,v 1.89 2008/03/07 10:02:29 ingo Exp
    mfile:
-----
```

The example code above creates a Hanning window object with 1000 points. The call to the AO constructor then creates a cdata type AO with 1000 points. This AO can then be multiplied against other AOs in order to window the data.

Creating AOs from waveform descriptions

MATLAB contains various functions for creating different waveforms, for example, square, sawtooth. Some of these functions can be called upon to create Analysis Objects. The following code creates an AO with a sawtooth waveform:

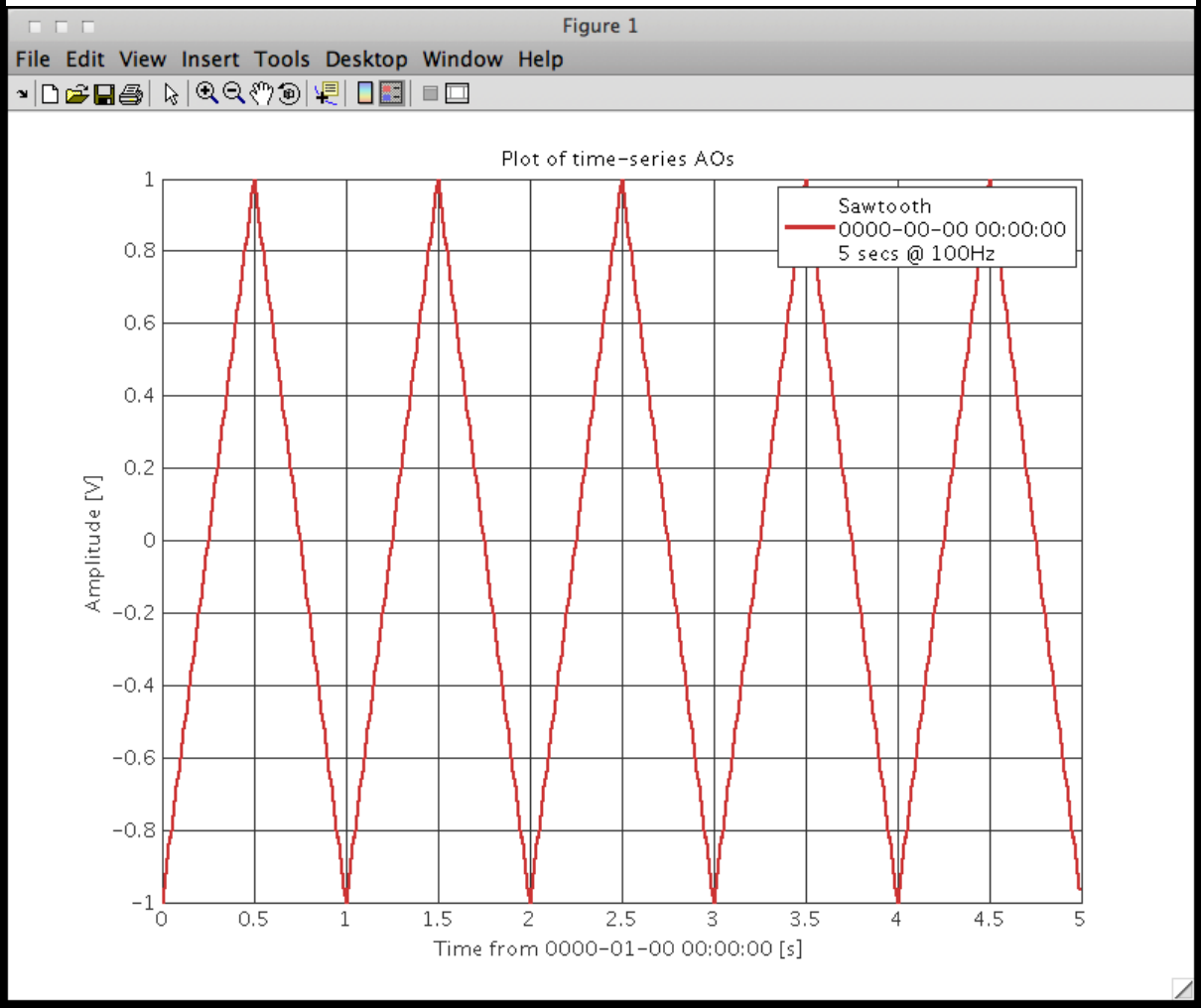
```
pl = plist();
pl = append(pl, 'fs', 100);
pl = append(pl, 'nsecs', 5);
pl = append(pl, 'waveform', 'Sawtooth');
pl = append(pl, 'f', 1);
pl = append(pl, 'width', 0.5);

asaw = ao(pl)
----- ao: asaw -----

      tag: -00001
      name: Sawtooth
provenance: created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
(R2007a) at 2007-06-24 10:37:51
  comment:
    data: tsdata / sawtooth(2*pi*1*t,0.5)
    hist: history / ao / $Id: ao.m,v 1.89 2008/03/07 10:02:29 ingo Exp
    mfile:
-----
```

You can call the `ipplot` function to view the resulting waveform:

```
ipplot(asaw);
```



◀ Analysis Objects

Converting existing data into Analysis Objects ▶

©LTP Team

Converting existing data into Analysis Objects

In some cases it is desirable to build AOs 'by hand' from existing data files. If the data file doesn't conform to one of the existing AO constructors, then a conversion script can easily be written to create AOs from your data files.

The following example shows how to convert an ASCII data file into Analysis Objects. The data file has 4 columns representing 4 different data channels. All 4 channels are sampled at the same rate of 10Hz. The conversion function returns 4 AOs.

```
function b = myConverter(filename, fs)

% MYCONVERTER converts a multi-column ASCII data file into multiple AOs.
%
% usage: b = myConverter(filename, fs)
%
% Inputs:
%   filename - name of input data file
%   fs       - sample rate of input data
%
% Outputs:
%   b        - vector of AOs
%
%
% Load the data from text file
data_in = load(filename);
ncols   = size(data_in, 2);

% Create AOs
b = [];
for j=1:ncols

% Get this column of data
d = data_in(:,j);

% name for this data
dataName = ['column' num2str(j)];

% Make tsdata object
ts = tsdata(d, fs);
% set name of data object
ts = set(ts, 'name', dataName);
% set xunits to seconds
ts = set(ts, 'xunits', 's');
% set xunits to Volts
ts = set(ts, 'yunits', 'V');

% make history object
h = history('myConverter', '0.1', plist(param('filename', filename)));

% make AO
a = ao(ts, h);

% set AO name
a = set(a, 'name', dataName);

% add to output vector
b = [b a];

end
```

The script works by loading the four columns of data from the ASCII file into a matrix. The script then loops over the number of columns and creates an AO for each column.

First a time-series (`tsdata`) object is made from the data in a column and given the input sample rate of the data. The 'name', 'xunits', and 'yunits' fields of the time-series data object are then set using calls to the `set` method of the `tsdata` class.

Next a history object is made with its 'name' set to 'myConverter', the version set to '0.1', and the input filename is recorded in a parameter list attached to the history object.

Following the creation of the data object and the history object, we can then create an Analysis Object. The name of the AO is then set and the object is added to a vector that is output by the function.

◀ Creating Analysis Objects

Saving Analysis Objects ▶

©LTP Team

Saving Analysis Objects

Analysis Objects can be saved to disk as either MATLAB binary files (.MAT) or as XML files (.XML). The following code shows how to do this:

```
save(aout, 'a.mat') % save AO aout to a .MAT file
save(aout, 'a.xml') % save AO aout to a .XML file
```


Plotting Analysis Objects

The data in an AO can be plotted using two different functions.

1. [A simple plot function `plot`](#)
2. [An intelligent plotting function `iplot`](#)

AO `plot` method

There are various calls to the simple `plot` method:

```
>> plot(a1)
```

Plots the data contained in the AO (`a1`) into the currently active axes.

As with the standard MATLAB `plot` function, you can pass line specifications to `ao/plot`. For example,

```
>> plot(a1, 'LineStyle', '--')
```

Plots the data contained in the AO (`a1`) into the currently active axes with a dashed line.

See `help plot` for further line specifications.

```
>> plot(axh, a1)
```

Plots the data contained in the AO (`a1`) into the axes specified by the given handle, `axh`.

You can get handles to the various plot elements as follows:

```
objects      >> line_h      = plot(a1)    % returns a handle to the line
axes objects >> [line_h, axes_h] = plot(a1)    % returns a handle to the line and
axes and figure objects >> [line_h, axes_h, figure_h] = plot(a1) % returns a handle to the line,
```

AO `iplot` method

The `iplot` method provides a more advanced plotting interface for AOs which tries to make good use of all the information contained within the input AOs. For example, if the `xunits` and `yunits` fields of the input AOs are set, these labels are used on the plot labels.

In addition, `iplot` can be configured using a input `plist`. The following examples show some of the possible ways to use `iplot`

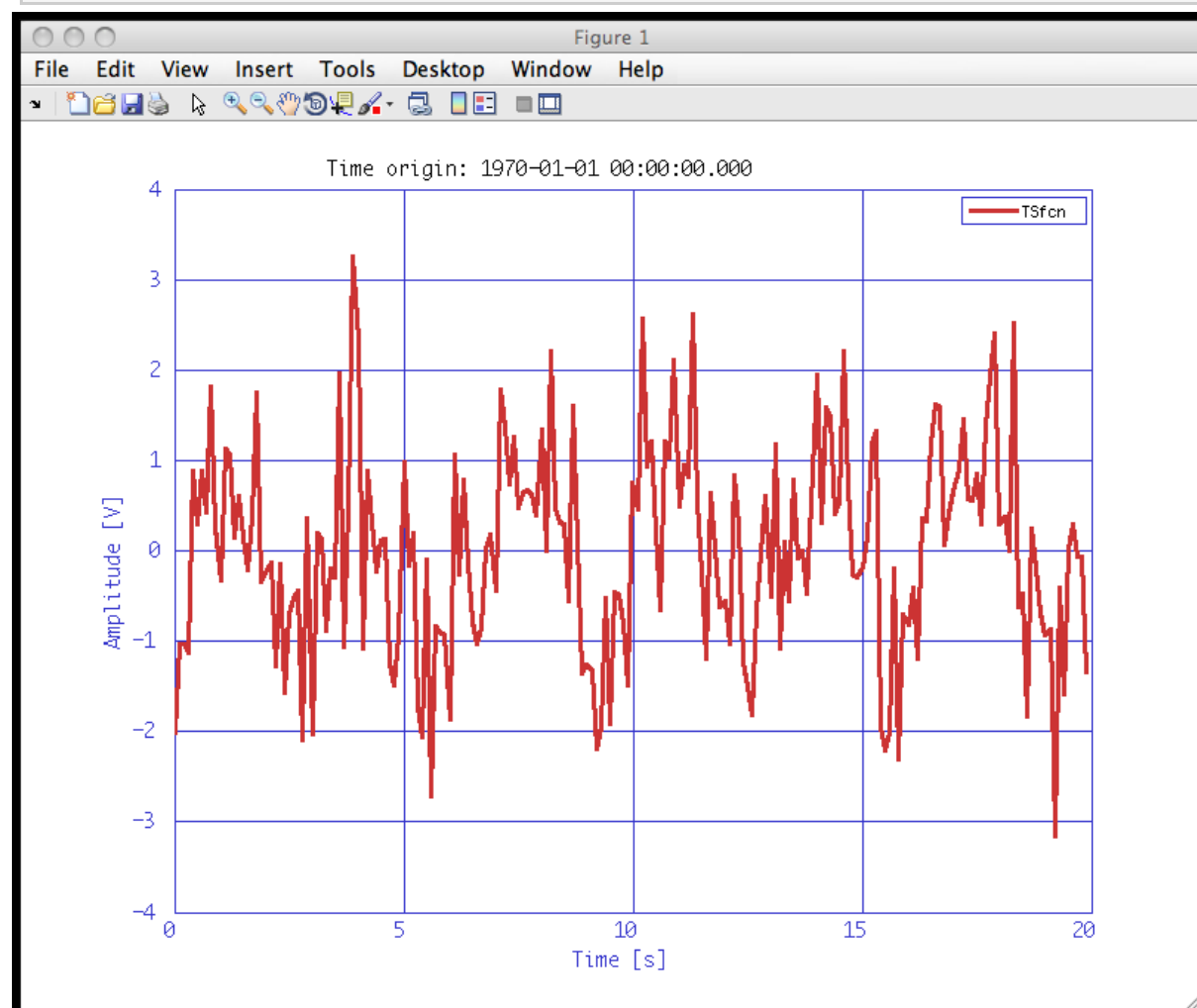
```
>> a1 = ao(plist('tsfcn', 'sin(2*pi*0.3*t) + randn(size(t))', 'fs', 10,
'nsecs', 20))
----- ao: a1 -----
      name: TSfcn
  provenance: created by hewitson@bobmac-2.local[172.16.251.1] on MACI/7.6
(R2008a Prerelease)/0.99 (R2008a Prerelease) at 2008-02-29 18:54:12.127
  description:
    data: tsdata / sin(2*pi*0.3*t) + randn(size(t)) [200x1] | (0, -
2.00888) (0.1, -1.02877) (0.2, -1.02874) (0.3, -1.13014) (0.4, 0.883107) ...
    hist: history / ao / $Id: ao.m,v 1.89 2008/03/07 10:02:29 ingo Exp
  mfilename:
  mdlfilename:
-----

>> a1.data
----- tsdata 01 -----
      name: sin(2*pi*0.3*t) + randn(size(t))
       fs: 10
       x: [200 1], double
       y: [200 1], double
  xunits: s
  yunits: V
   nsecs: 20
    t0: 1970-01-01 00:00:00.000
-----
```

Creates a time-series AO. If we look at the data object contained in this AO, we see that the xunits and yunits are set to the defaults of seconds [s] and Volts [V].

If we plot this object with `ipplot` we see these units reflected in the x and y axis labels.

```
>> ipplot(a1)
```



We also see that the time-origin of the data (`t0` field of the `tsdata` class) is displayed as the plot

title.

©LTP Team

Parameter Lists

Any algorithm that requires input parameters to configure its behaviour should take a Parameter List ([plist](#)) object as input. A `plist` object contains a vector of Parameter ([param](#)) objects.

The following sections introduce parameters and parameter lists:

- [Creating Parameters](#)
- [Creating lists of Parameters](#)
- [Working with Parameter Lists](#)

Creating Parameters

Parameter objects are used in the LTPDA Toolbox to configure the behaviour of algorithms. A parameter (`param`) object has two main properties:

- 'key' — The parameter name
- 'val' — The parameter value

See [param class](#) for further details. The 'key' property is always stored in upper case. The 'value' of a parameter can be any LTPDA object, as well as most standard MATLAB types.

Parameter values can take any form: vectors or matrices of numbers; strings; other objects, for example a `specwin` (spectral window) object.

Parameters are created using the `param` class constructor. The following code shows how to create a parameter 'a' with a value of 1

```
>> p = param('a', 1)
---- param 1 ----
key: a
val: 1
-----
```

The contents of a parameter object can be accessed as follows:

```
>> key = p.key; % get the parameter key
>> val = p.val; % get the parameter value
```


Creating lists of Parameters

Parameters can be grouped together into parameter lists (plist).

- [Creating parameter lists from parameters](#)
- [Creating parameter lists directly](#)
- [Appending parameters to a parameter list](#)
- [Finding parameters in a parameter list](#)
- [Removing parameters from a parameter list](#)
- [Setting parameters in a parameter list](#)
- [Combining multiple parameter lists](#)

Creating parameter lists from parameters.

The following code shows how to create a parameter list from individual parameters.

```
>> p1 = param('a', 1); % create first parameter
>> p2 = param('b', specwin('Hanning', 100)); % create second parameter
>> pl = plist([p1 p2]) % create parameter list
----- plist 01 -----
n params: 2
---- param 1 ----
key: A
val: 1
-----
---- param 2 ----
key: B
val: specwin
----- Hanning -----

alpha: 0
psll: 31.5
rov: 50
nenbw: 1.5
w3db: 1.4382
flatness: -1.4236
ws: 50
ws2: 37.5
win: 100

-----

-----
-----
```

Creating parameter lists directly.

You can also create parameter lists directly using the following constructor format:

```
>> pl = plist('a', 1, 'b', 'hello')
----- plist 01 -----
n params: 2
---- param 1 ----
key: A
val: 1
-----
---- param 2 ----
key: B
val: 'hello'
-----
-----
```

Appending parameters to a parameter list.

Additional parameters can be appended to an existing parameter list using the `append` method:

```
>> pl = append(pl, param('c', 3)) % append a third parameter
----- plist 01 -----
n params: 3
---- param 1 ----
key: A
val: 1
-----
---- param 2 ----
key: B
val: 'hello'
-----
---- param 3 ----
key: C
val: 3
-----
-----
```

Finding parameters in a parameter list.

Accessing the contents of a `plist` can be achieved in two ways:

```
>> p1 = pl.params(1); % get the first parameter
>> val = find(pl, 'b'); % get the second parameter
```

If the parameter name ('key') is known, then you can use the `find` method to directly retrieve the value of that parameter.

Removing parameters from a parameter list.

You can also remove parameters from a parameter list:

```
>> pl = remove(pl, 2) % Remove the 2nd parameter in the list
>> pl = remove(pl, 'a') % Remove the parameter with the key 'a'
```

Setting parameters in a parameter list.

You can also set parameters contained in a parameter list:

```
>> pl = plist('a', 1, 'b', 'hello')
>> pl = pset(pl, 'a', 5, 'b', 'ola'); % Change the values of the parameter with the
keys 'a' and 'b'
```

Combining multiple parameter lists.

Parameter lists can be combined:

```
>> pl = combine(pl1, pl2)
```

If `pl1` and `pl2` contain a parameter with the same key name, the output `plist` contains a parameter with that name but with the value from the first parameter list input.



©LTP Team

Spectral Windows

Spectral windows are an essential part of any spectral analysis. As such, great care has been taken to implement a complete and accurate set of window functions. The window functions are implemented as a class `specwin`. The properties of the class are given in [specwin class](#).

The following pages describe the implementation of spectral windows in the LTPDA framework:

- [What are LTPDA spectral windows?](#)
- [Creating spectral windows](#)
- [Visualising spectral windows](#)
- [Using spectral windows](#)

What are LTPDA spectral windows?

MATLAB already contains a number of window functions suitable for spectral analysis. However, these functions simply return vectors of window samples; no additional information is given. It is also desirable to have more information about a window function, for example, its normalised equivalent noise bandwidth (NENBW), its peak side-lobe level (PSLL), and its recommended overlap (ROV).

The [specwin](#) class implements many window functions as class objects that contain many descriptive properties. The following table lists the available window functions and some of their properties:

Window name	NENBW	PSLL [dB]	ROV [%]
Rectangular	1.000	-13.3	0.0
Welch	1.200	-21.3	29.3
Bartlett	1.333	-26.5	50.0
Hanning	1.500	-31.5	50.0
Hamming	1.363	-42.7	50.0
Nuttall3	1.944	-46.7	64.7
Nuttall4	2.310	-60.9	70.5
Nuttall3a	1.772	-64.2	61.2
Nuttall3b	1.704	-71.5	59.8
Nuttall4a	2.125	-82.6	68.0
Nuttall4b	2.021	-93.3	66.3
Nuttall4c	1.976	-98.1	65.6
BH92	2.004	-92.0	66.1
SFT3F	3.168	-31.7	66.7
SFT3M	2.945	-44.2	65.5

FTNI	2.966	-44.4	65.6
SFT4F	3.797	-44.7	75.0
SFT5F	4.341	-57.3	78.5
SFT4M	3.387	-66.5	72.1
FTHP	3.428	-70.4	72.3
HFT70	3.413	-70.4	72.2
FTSRS	3.770	-76.6	75.4
SFT5M	3.885	-89.9	76.0
HFT90D	3.883	-90.2	76.0
HFT95	3.811	-95.0	75.6
HFT116D	4.219	-116.8	78.2
HFT144D	4.539	-114.1	79.9
HFT169D	4.835	-169.5	81.2
HFT196D	5.113	-196.2	82.3
HFT223D	5.389	-223.0	83.3
HFT248D	5.651	-248.0	84.1

In addition to these 'standard' windows, Kaiser windows can be designed to give a chosen PSL.

◀ Spectral Windows

Create spectral windows ▶

Create spectral windows

To create a spectral window object, you call the `specwin` class constructor. The following code fragment creates a 100-point Hanning window:

```
>> w = specwin('Hanning', 100)
----- Hanning -----
alpha: 0
psll: 31.5
rov: 50
nenbw: 1.5
w3db: 1.4382
flatness: -1.4236
ws: 50
ws2: 37.5
win: 100
-----
```

[List of available window functions](#)

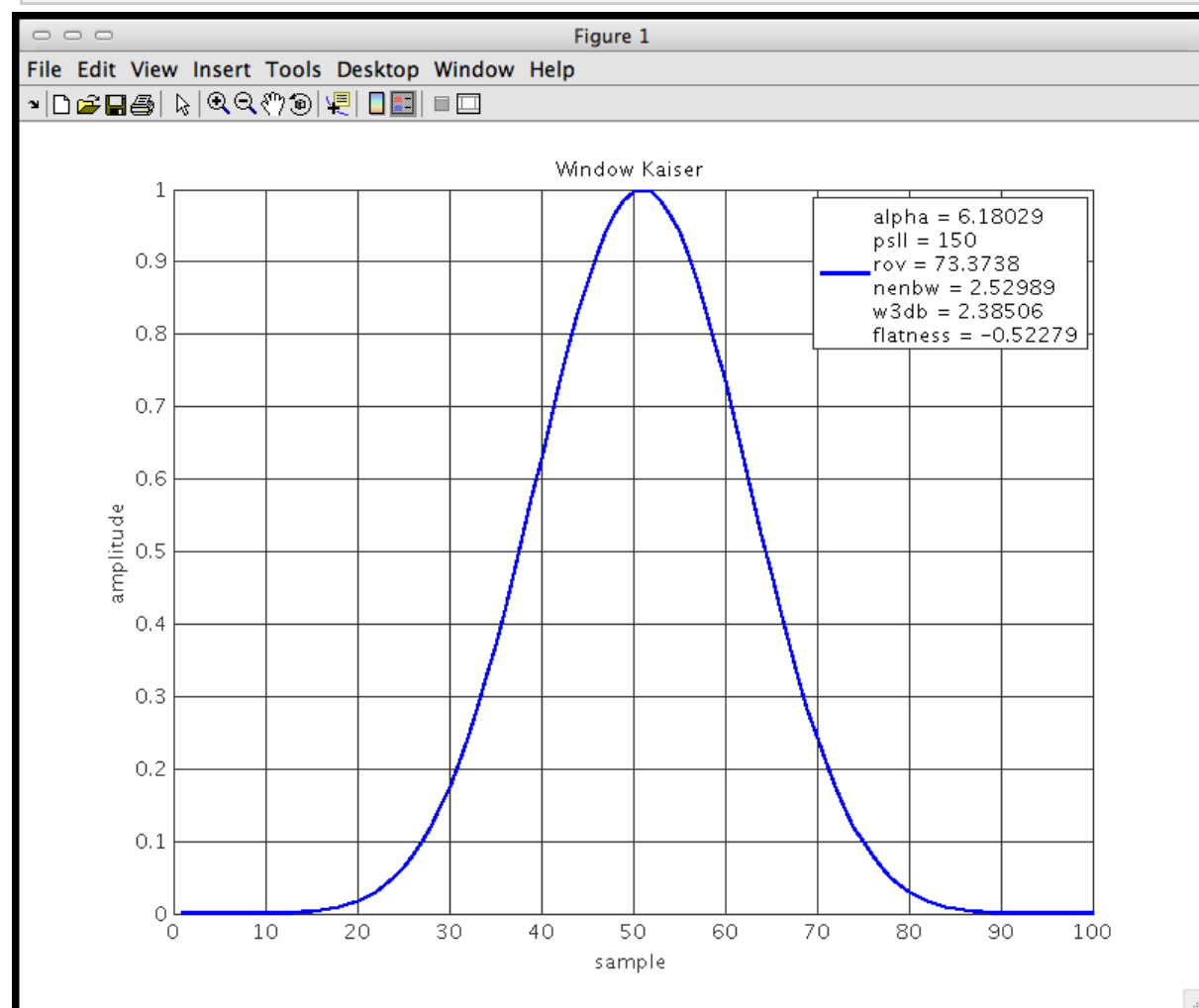
In the special case of creating a Kaiser window, the additional input parameter, PSLL, must be supplied. For example, the following code creates a 100-point Kaiser window with -150dB peak side-lobe level:

```
>> w = specwin('Kaiser', 100, 150)
----- Kaiser -----
alpha: 6.18029
psll: 150
rov: 73.3738
nenbw: 2.52989
w3db: 2.38506
flatness: -0.52279
ws: 28.2558
ws2: 20.1819
win: 100
-----
```


Visualising spectral windows

The `specwin` class has a `plot` method which will plot the response of the given window function in the current Figure:

```
w = specwin('Kaiser', 100, 150);  
figure  
plot(w)
```



Windows can also be visualised using the [Spectral Window Viewer](#).

◀ Create spectral windows

Using spectral windows ▶

Using spectral windows

Spectral windows are typically used in spectral analysis algorithms. In all LTPDA spectral analysis functions, spectral windows are specified as parameters in an input parameter list. The following code fragment shows the use of `ltpda_pwelch` to estimate an Amplitude Spectral Density of the time-series captured in the input AO, `a_in`. The help for `ltpda_pwelch` reveals that the required parameter for setting the window function is 'Win'.

```
w = specwin('Kaiser', 100, 150);  
pl = plist(param('Win', w))  
axx = ltpda_pwelch(a_in, pl);
```

In this case, the size of the spectral window (number of samples) may not match the length of the segments in the spectral estimation. The `ltpda_pwelch` algorithm then recomputes the window using the input design but for the correct length of window function.

Spectral windows can also be used more directly by first converting them to Analysis Objects. The following code fragment converts a `specwin` object to an Analysis Object. This AO is then multiplied against another time-series AO to window the data.

```
a = ao('data1.txt')  
w = specwin('Kaiser', len(a), 150);  
wa = ao(w);  
a_win = a.*wa;
```

Note here that the `len` method of the AO class is used to produce a window function that is of the same length as the time-series contained in `a`.



Simulation/modelling

Content needs written...

◀ Using spectral windows

Generating model noise ▶

©LTP Team

Generating model noise

Generating non-white random noise means producing arbitrary long time series with a given spectral density. Such time series are needed for example for the following purposes:

- To generate test data sets for programs that compute spectral densities,
- as inputs for various simulations.

One way of doing this is to apply digital filters (FIR or IIR) to white input noise. This approach is often used because of its simplicity but it has some disadvantages: For complicated spectra a matching filter is not trivial to find. The filter has to be split into several simple sections. Via nonlinear optimization techniques optimized filters can be found. Those techniques however are non deterministic such that their convergence is not guaranteed. So although the resulting filter transfer function is well known, it may not perfectly match the given spectrum. Moreover the filtering method requires a 'warm-up period'

A different approach is implemented in LTPDA as [Franklin noise-generator](#). It produces spectral densities according to a given pole zero model (see [Pole/Zero Modeling](#)) and does not require any warm-up period.

Franklin noise-generator

The following sections gives an introduction to the [generation of model noise](#) using the noise generator implemented in LTPDA.

- [Franklin's noise generator](#)
- [Description](#)
- [Call](#)
- [Inputs](#)
- [Outputs](#)
- [Usage](#)
- [Case 1: Starting from a given pole/zero model](#)
- [Case 2: Starting from given matrices](#)
- [Case 3: Starting from a given state vector and matrices](#)

Franklin's noise generator

Franklin's noise generator is a method to generate arbitrarily long time series with a prescribed spectral density. The algorithm is based on the paper 'Numerical simulation of stationary and non-stationary gaussian random processes' by Franklin, Joel N. (SIAM review, Volume 7, Issue 1, page 68-80, 1956)

See [Generating model noise](#) for more general information on this.

Franklin's method does not require any 'warm up' period. It starts with a transfer function given as ratio of two polynomials.

The generator operates on a real state vector y of length n which is maintained between invocations. It produces samples of the time series in equidistant steps $\tau = 1/f_s$, where f_s is the sampling frequency.

- $y_0 = T_{init} * r$, on initialization
- $y_i = E * y_{i-1} + T_{prop} * r$, to propagate
- $x_i = a * y_i$, the sampled time series.

r is a vector of independent normal Gaussian random numbers T_{init} , E , T_{prop} which are real matrices and a which is a real vector are determined once by the algorithm.

Description

[ltpda_noisegen](#) uses `pzm2ab.m` and `franklin.m` to generate a time-series from a given `pzmodel`

Call

```
>> [b, pl1, pl2] = ltpda_noisegen(pl)
>> [b, pl1] = ltpda_noisegen(pl)
>> b = ltpda_noisegen(pl1, pl2)
```

Inputs

for the first function call the parameter list has to contain at least:

- `nsecs` – number of seconds (length of time series)

- f_s – sampling frequency
- pzmodel with gain

Outputs

- b – analysis object containing the resulting time series
- $pl1$ – parameter list containing the last state vector y .
- $pl2$ – parameter list containing the following parameters:
 - T_{init} – matrix to calculate initial state vector
 - T_{prop} – matrix to calculate propagation vector
 - E – matrix to calculate propagation vector
 - num – numerator coefficients
 - den – denominator coefficients

Usage

[ltpda_noisegen](#) consists of the following four main functions.

- [ngconv](#)
- [ngsetup](#)
- [nginit](#)
- [ngprop](#)

Depending on the [Inputs](#) different functions are called by [ltpda_noisegen](#).

First a parameter list of the input parameters is to be done. For further information on this look at [Creating parameter lists from parameters](#).

Case 1: Starting from a given pole/zero model

The most common case would be to start from a pole zero model pzm , the number of seconds the resulting time series should have $nsecs$ and the sampling frequency f_s . In this case all of the four main functions mentioned above are called one after the other.

The function call can then look like this:

```
>> [b, pl1, pl2] = ltpda_noisegen(pl)
```

The output will be an analysis object b containing the time series and two parameter lists ($pl1$, $pl2$). The first ($pl1$) contains the state vector y that can be used as input to generate a second time series with the same characteristics. The second ($pl2$) contains:

- the matrices
 - T_{init}
 - T_{prop}
 - E
- the coefficients of transfer function:
 - num
 - den
- the input parameters
 - $gain$
 - f_s
 - $nsecs$

Case 2: Starting from given matrices

The matrices T_{init} , T_{prop} and E can be known from a previous calculation (i.e [ltpda_noisegen](#),

LISO).

These matrices can be stored as parameters of a parameter list and inputted into [ltpda_noisege](#).

In this case the function starts the operation with calling [nginit](#) first and finally [ngprop](#). The output will be the same as for [case 1](#).

Case 3: Starting from a given state vector and matrices

In this case a state vector y known from a previous run of the function is given as input together with the matrices.

[ltpda_noisege](#) will now only call [ngprop](#).

The output will again be the same as above.

◀ Generating model noise

Pole/Zero Modelling ▶

©LTP Team

Pole/Zero Modelling

Pole/zero modelling is implemented in the LTPDA Toolbox using three classes: a [pole class](#), a [zero class](#), and a pole/zero model class ([pzmodel class](#)).

The following pages introduce how to produce and use pole/zero models in the LTPDA environment.

- [Creating poles and zeros](#)
- [Building a model](#)
- [Model helper GUI](#)
- [Converting models to IIR filters](#)

Creating poles and zeros

Poles and zeros are treated the same with regards creation, so we will look here at poles only. The meaning of a pole and a zero only becomes important when creating a pole/zero model.

Poles are specified by in the LTPDA Toolbox by a frequency, f , and (optionally) a quality factor, Q .

The following code fragment creates a real pole at 1Hz:

```
>> p1 = pole(1)
---- pole 1 ----
real pole: 1 Hz
-----
```

To create a complex pole, you can specify a quality factor. For example,

```
>> p1 = pole(10, 4)
---- pole 1 ----
complex pole: 10 Hz, Q=4 [-0.0019894+0.015791i]
-----
```

creates a complex pole at 10Hz with a Q of 4. You can also see that the complex representation is also shown, but only one part of the conjugate pair.

Building a model

Poles and zeros can be combined together to create a pole/zero model. In addition to a list of poles and zeros, a gain factor can be specified such that the resulting model is of the form:

$$H(s) = G \frac{(s - z_1)(s - z_2) \dots}{(s - p_1)(s - p_2) \dots}$$

The following sections introduce how to produce and use pole/zero models in the LTPDA environment.

- [Direct form](#)
- [Creating from a plist](#)
- [Computing the response of the model](#)

Direct form

The following code fragment creates a pole/zero model consisting of 2 poles and 2 zeros with a gain factor of 10:

```
>> poles = [pole(1,2) pole(40)];
>> zeros = [zero(10,3) zero(100)];
>> pzm = pzmodel(10, poles, zeros)
---- pzmodel 1 ----
model: pzmodel
gain: 10
pole 001: pole(1,2)
pole 002: pole(40)
zero 001: zero(10,3)
zero 002: zero(100)
-----
```

Creating from a plist

You can also create a pzmodel by passing a parameter list. The following example shows this

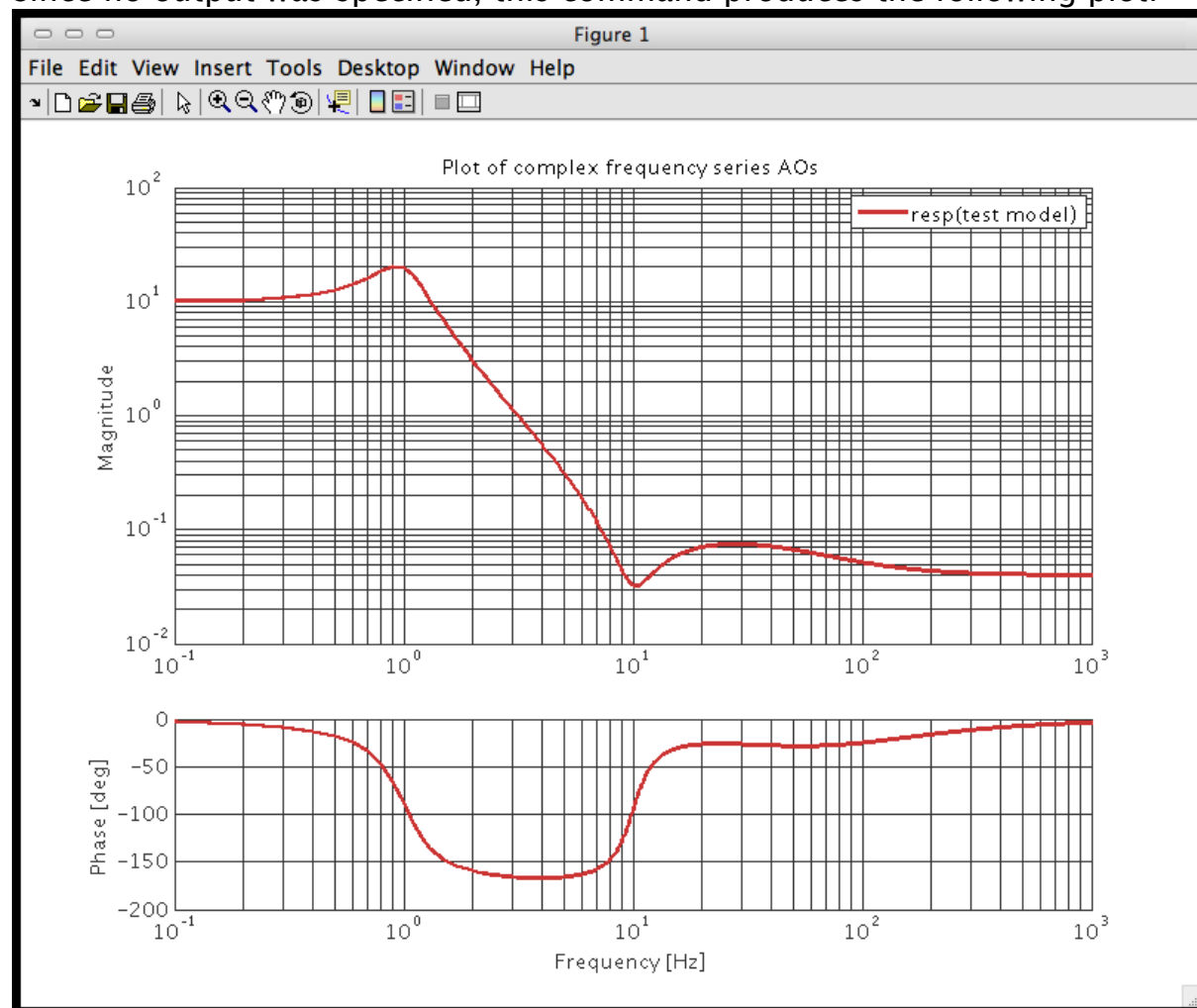
```
>> pl = plist('name', 'test model', ...
              'gain', 10, ...
              'poles', [pole(1,2) pole(40)], ...
              'zeros', [zero(10,3) zero(100)]);
>> pzm = pzmodel(pl)
---- pzmodel 1 ----
model: test model
gain: 10
pole 001: pole(1,2)
pole 002: pole(40)
zero 001: zero(10,3)
zero 002: zero(100)
-----
```

Computing the response of the model

The frequency response of the model can be generated using the `resp` method of the `pzmodel` class. To compute the response of the model created above:

```
>> resp(pzm)
```

Since no output was specified, this command produces the following plot:



You can also specify the frequency band over which to compute the response by passing a `plist` to the `resp` method, as follows:

```
>> rpl = plist('f1', 0.1, ...
              'f2', 1000, ...
              'nf', 10000);
>> a = resp(pzm, rpl)
----- ao: a -----
      tag: -00001
      name: resp(test model)
  provenance: created by unknown@42.104.179.202.in-addr.arpa[202.179.104.42] on
MACI/7.4 (R2007a)/0.3 (R2007a) at 2007-07-05 09:56:47
    comment:
      data: fsdata / resp(test model)
      hist: history / resp / $Id: resp.m,v 1.16 2008/02/24 21:43:59 mauro Exp
      mfile:
-----
```

In this case, the response is returned as an Analysis Object containing `fsdata`. You can now plot the AO using the `ipplot` function.

Model helper GUI

A simple GUI exists to help you build pole/zero models. To start the GUI, type

```
>> pzmodel_helper
```

More details on the [PZModel Helper GUI](#).

Converting models to IIR filters

Pole/zero models can be converted to IIR filters using the bilinear transform. The result of the conversion is an `miir` object. To convert a model, you call the constructor of the `miir` class with a `plist` input:

```
>> filt = miir(plist('pzmodel', pzm))
```

If no sample rate is specified, then the conversion is done for a sample rate equal to 8 times the highest pole or zero frequency. You can set the sample rate by specifying it in the parameter list:

```
>> filt = miir(plist('pzmodel', pzm, 'fs', 1000))
```

For more information of IIR filters in LTPDA, see [IIR Filters](#).

Signal Pre-processing in LTPDA

Signal pre-processing in LTPDA consists on a set of functions intended to pre-process data prior to further analysis. Pre-processing tools are focused on data sampling rates manipulation, data interpolation, spike cleaning and gap filling functions.

The following pages describe the different pre-processing tools available in the LTPDA toolbox:

- [Downsampling data](#)
- [Upsampling data](#)
- [Resampling data](#)
- [Interpolating data](#)
- [Spikes reduction in data](#)
- [Data gap filling](#)

Downsampling data

Downsampling is the process of reducing the sampling rate of a signal. [Downsample](#) reduces the sampling rate of the input AOs by an integer factor by picking up one out of N samples. Note that no anti-aliasing filter is applied to the original data. Moreover, a `offset` can be specified, i.e., the sample at which the output data starts ---see examples below.

Syntaxis

```
b = downsample(a, pl)
```

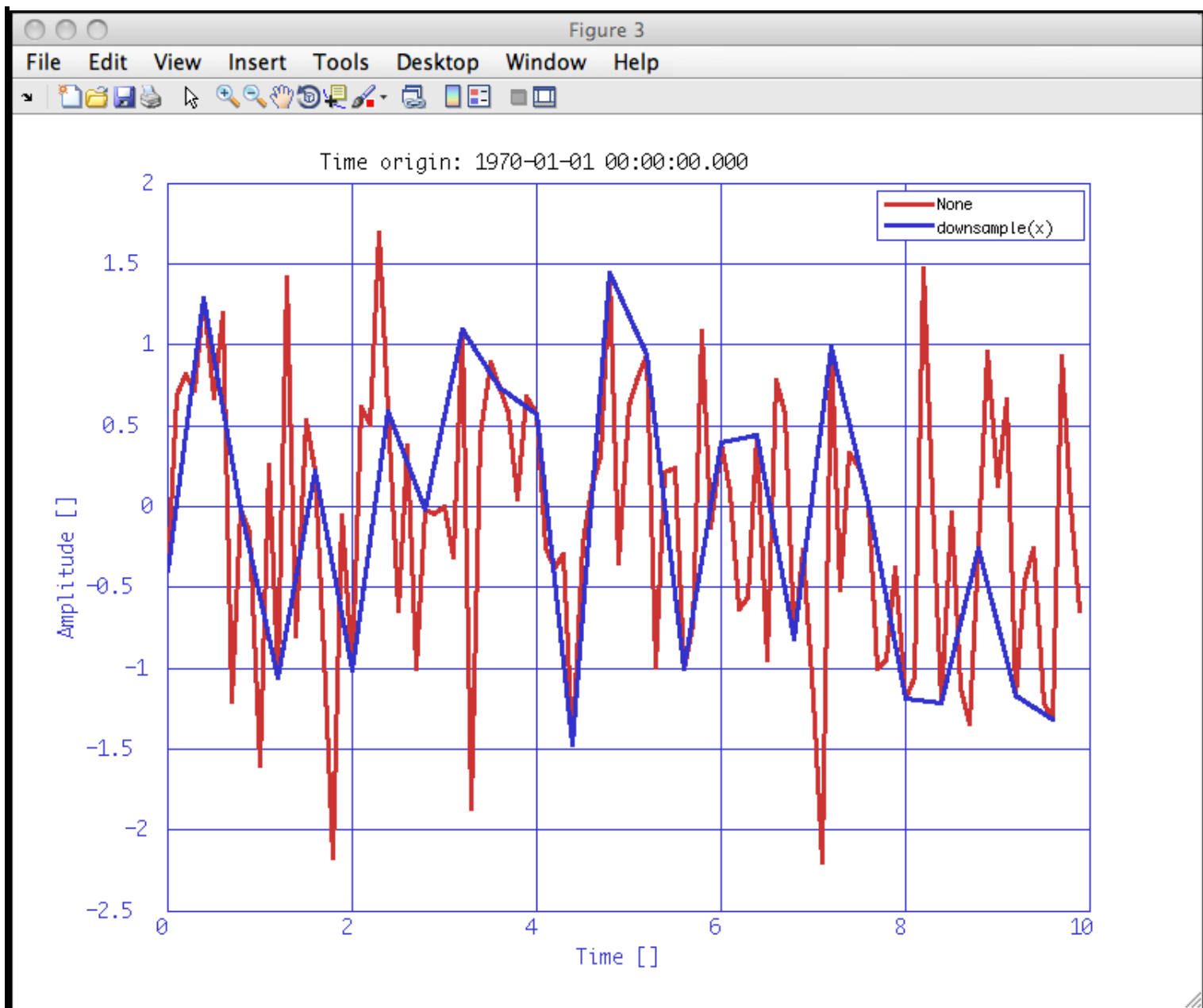
With the following parameters:

- `'factor'` – decimation factor [by default is 1: no downsampling] (must be an integer)
- `'offset'` – sample offset for decimation

Examples

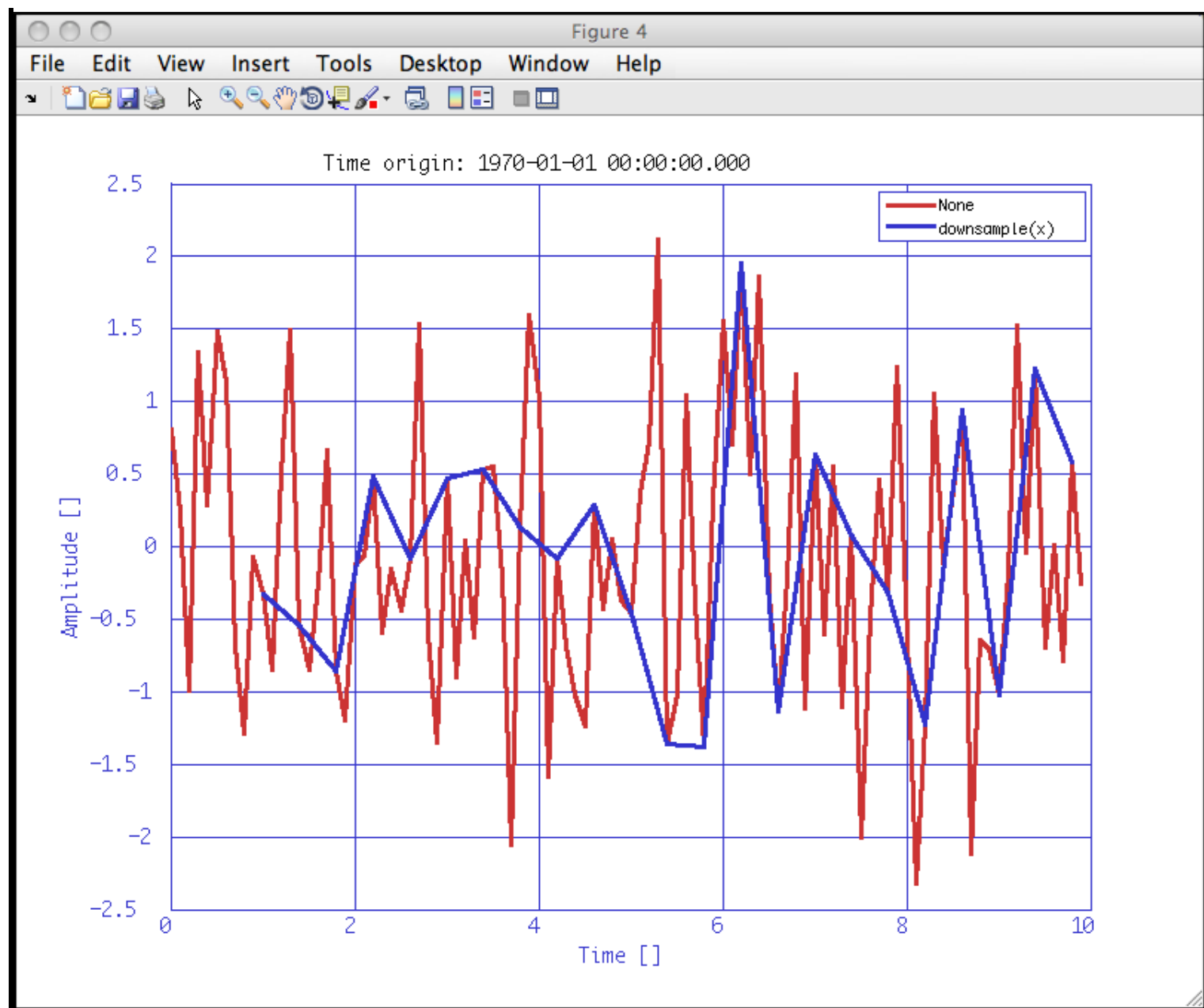
1. Downsampling a sequence of random data at original sampling rate of 10 Hz by a factor of 4 (`fsout` = 2.5 Hz) and no `offset`.

```
x=ao(tsddata(randn(100,1),10)); % create an AO of random data with fs = 10 Hz
pl = plist(); % create an empty parameters list
pl = append(pl, param('factor', 4)); % add the decimation factor
y = downsample(x, pl); % downsample the input AO, x
iplot(x, y) % plot original,x, and decimated,y, AOs
```



2. Downsampling a sequence of random data at original sampling rate of 10 Hz by a factor of 4 ($f_{\text{out}} = 2.5$ Hz) and offset = 10.

```
x=ao(tsdata(randn(100,1),10)); % create an A0 of random data with fs = 10 Hz.
pl = plist(); % create an empty parameters list
pl = append(pl, param('factor', 4)); % add the decimation factor
pl = append(pl, param('offset', 10)); % add the offset parameter
y = downsample(x, pl); % downsample the input A0, x
iplot(x, y) % plot original, x, and decimated, y, A0s
```



◀ Signal Pre-processing in LTPDA

Upsampling data ▶

©LTP Team

Upsampling data

Upsampling is the process of increasing the sampling rate of a signal. [Upsample](#) increases the sampling rate of the input AOs by an integer factor. LTPDA [upsample](#) overloads `upsample` function from Matlab Signal Processing Toolbox. This function increases the sampling rate of a signal by inserting $(n-1)$ zeros between samples. The upsampled output has $(n \cdot \text{input})$ samples. In addition, an initial phase can be specified and, thus, a delayed output of the input can be obtained by using this option.

Syntaxis

```
b = upsample(a, pl)
```

With the following parameters:

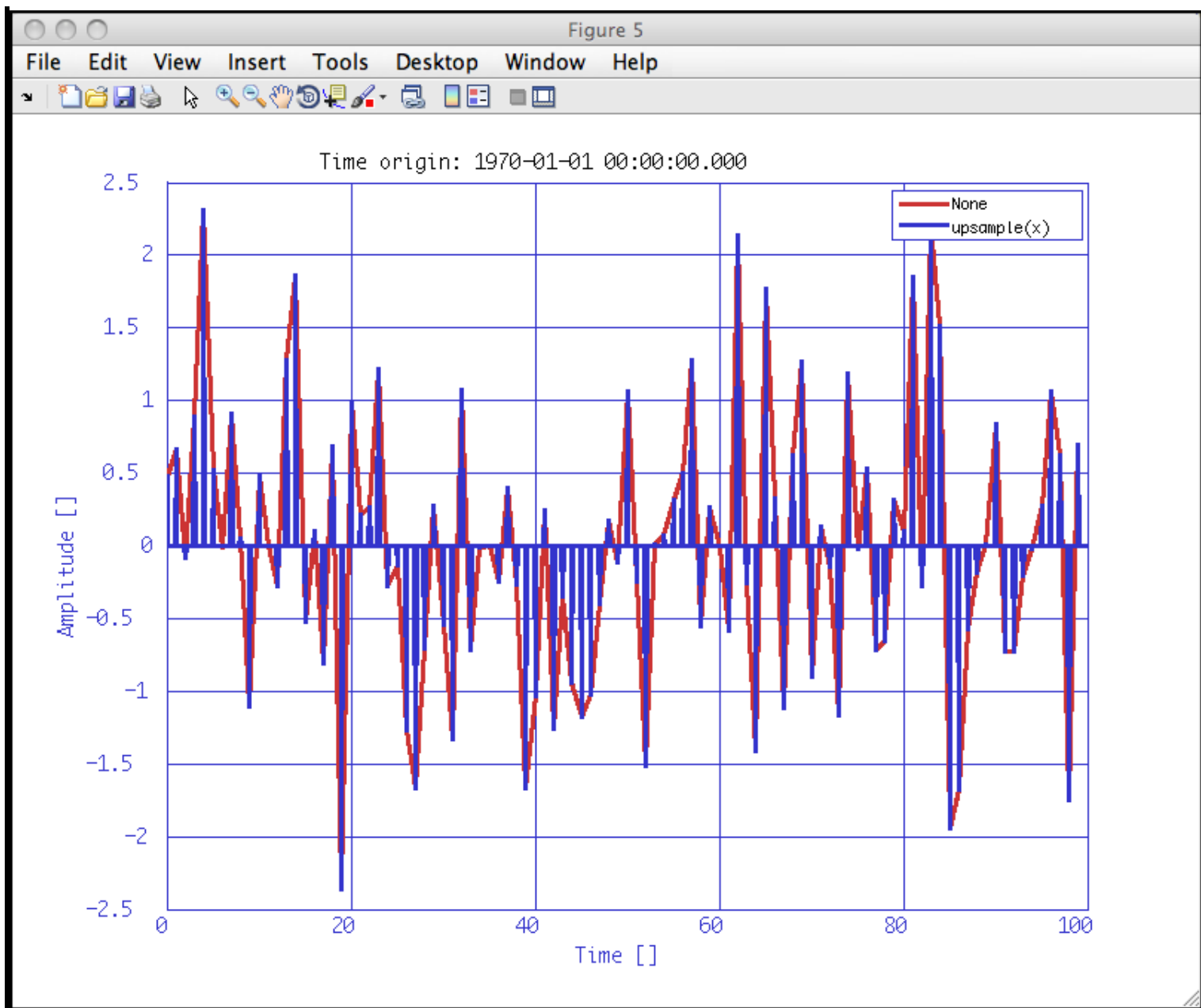
- 'N' – specify the desired upsample rate
- 'phase' – specify an initial phase range $[0, N-1]$

Examples

1. Upsampling a sequence of random data at original sampling rate of 1 Hz by a factor of 10 with no initial phase.

10

```
x=ao(tsddata(randn(100,1),1)); % create an AO of random data sampled at 1 Hz.  
pl = plist(); % create an empty parameters list  
pl = append(pl, param('N', 10)); % increase the sampling frequency by a factor of  
  
y = upsample(x, pl); % resample the input AO (x) to obtain the upsampled AO (y)  
iplot(x, y) % plot original and upsampled data
```



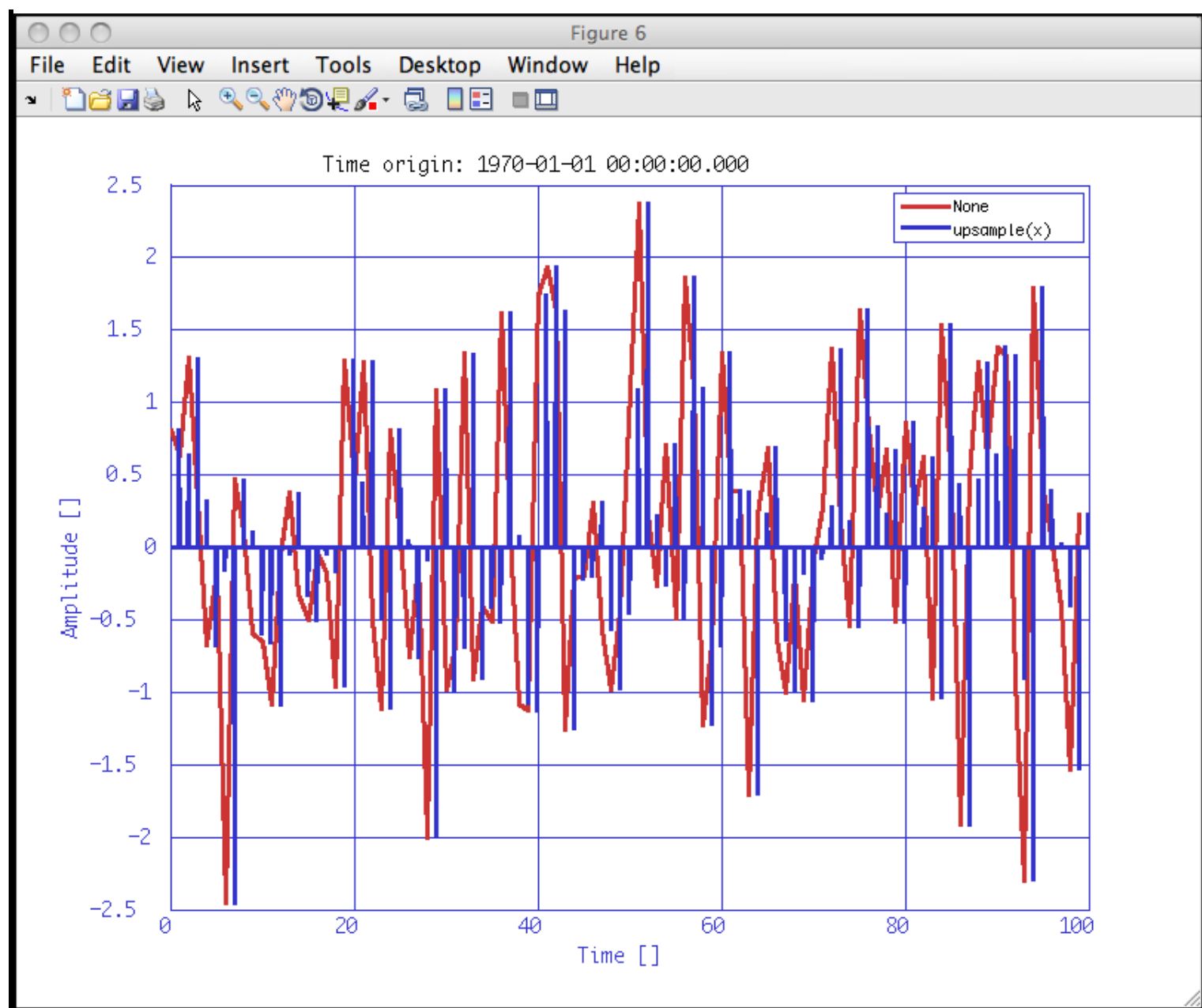
2. Upsampling a sequence of random data at original sampling rate of 1 Hz by a factor of 21 with a phase of 20 samples.

21

data

delayed AO (y)

```
x=ao(tsddata(randn(100,1),1)); % create an AO of random data sampled at 1 Hz.
pl = plist(); % create an empty parameters list
pl = append(pl, param('N', 21)); % increase the sampling frequency by a factor of
pl = append(pl, param('phase', 20)); % add phase of 20 samples to the upsampled
y = upsample(x, pl); % resample the input AO (x) to obtain the upsampled and
ipplot(x, y) % plot original and upsampled data
```



◀ Downsampling data

Resampling data ▶

©LTP Team

Resampling data

Resampling is the process of changing the sampling rate of data. [Resample](#) changes the sampling rate of the input AOs to the desired output sampling frequency. LTPDA [resample](#) overloads `resample` function of Matlab Signal Processing Toolbox for AOs.

```
b = resample(a, pl)
```

With the following parameters:

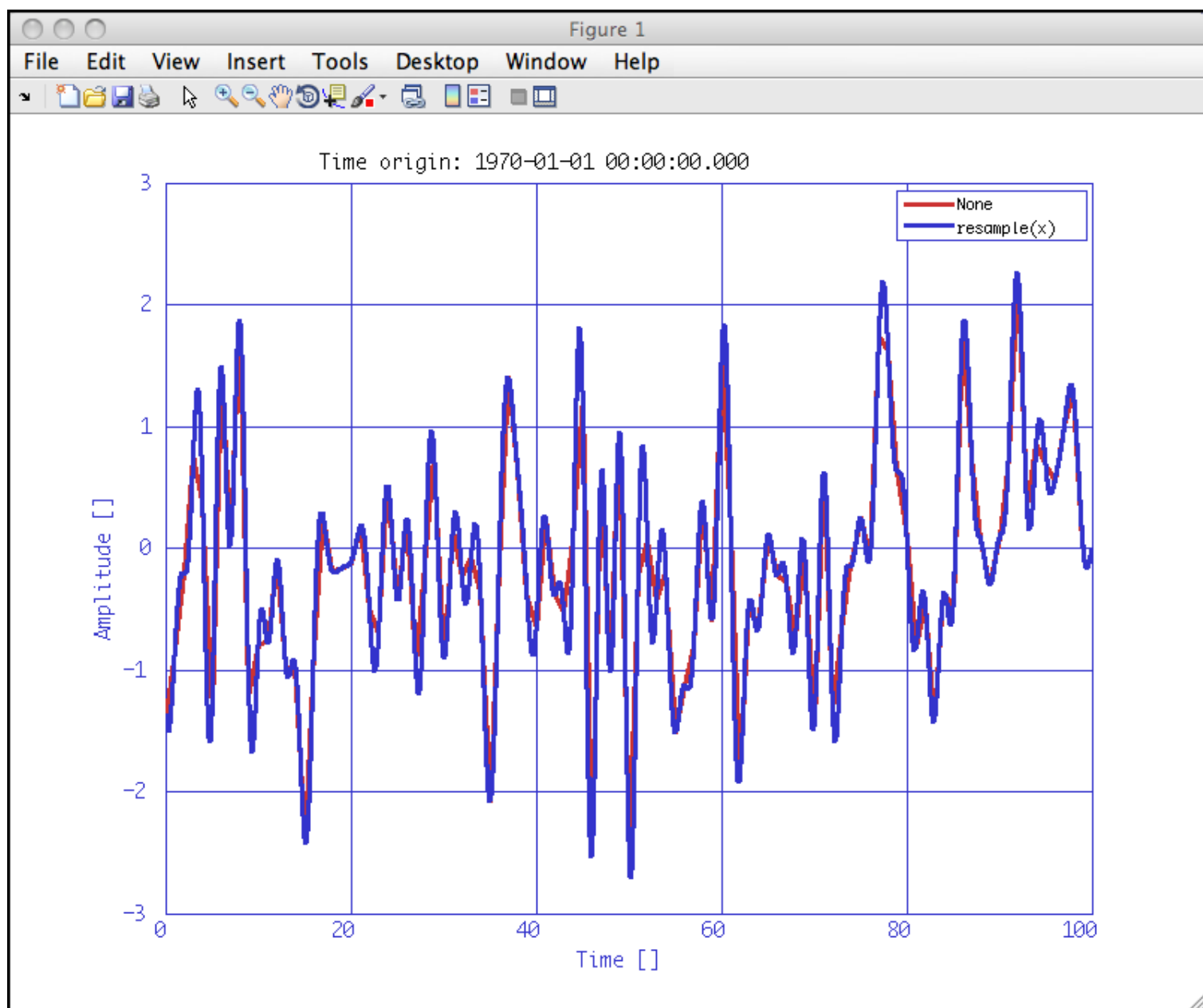
- 'fsout' – specify the desired output frequency (must be positive and integer)
- 'filter' – specified filter applied to the input, a, in the resampling process

Examples

1. Resampling a sequence of random data at original sampling rate of 1 Hz at an output sampling of 50 Hz.

(y)

```
x=ao(tsddata(randn(100,1),1)); % create AO of random data with fs = 1 Hz.  
pl = plist(); % create an empty parameters list  
pl = append(pl, param('fsout', 50)); % add fsout = 50 Hz to parameters list  
y = resample(x, pl); % resample the input AO (x) to obtain the resampled output AO  
iplot(x, y) % plot original and resampled data
```

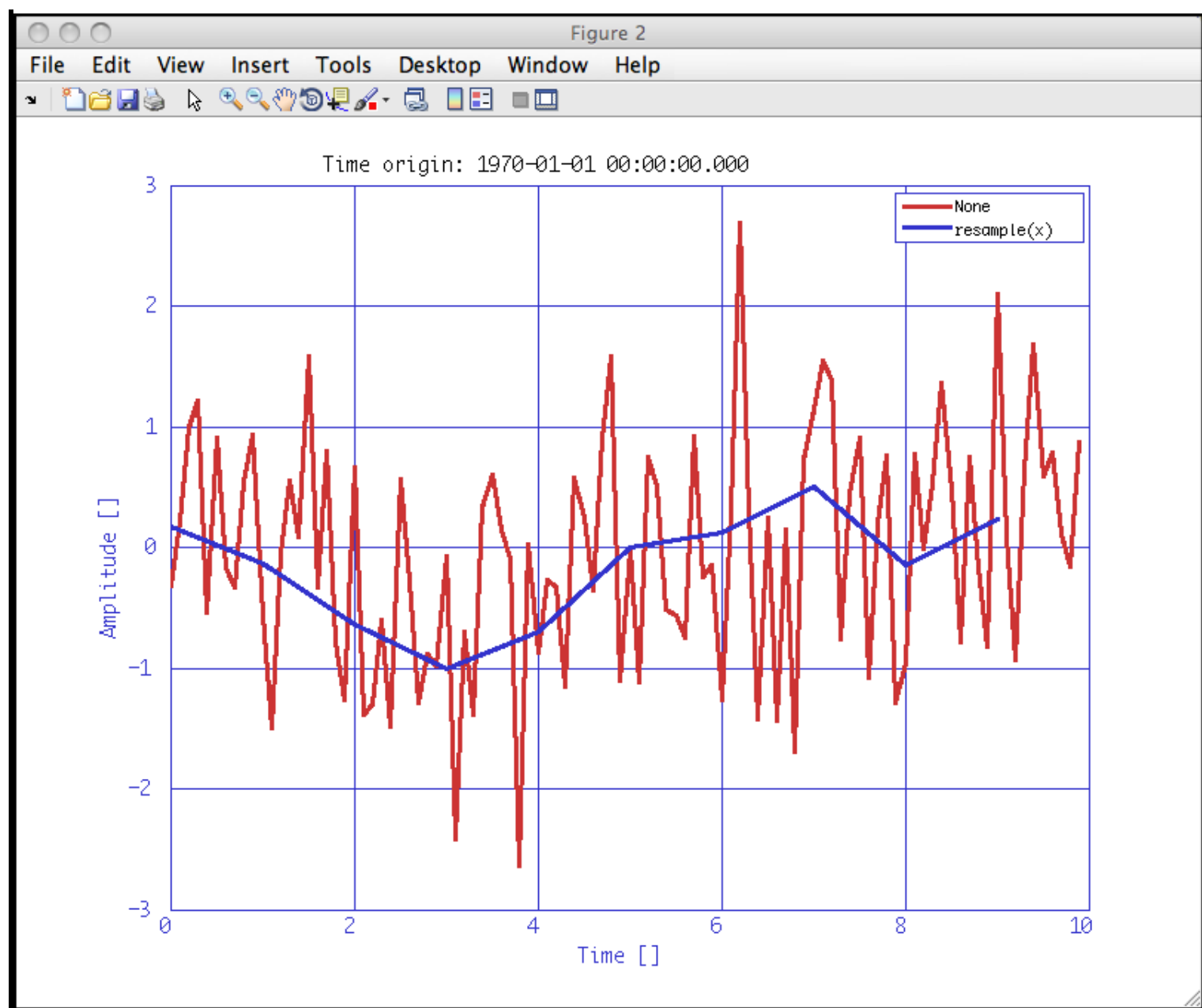



1. Resampling a sequence of random data at original sampling rate of 10 Hz at an output sampling of 1 Hz with a filter defined by the user.

```
x=ao(tsddata(randn(100,1),10)); % create AO of random data with fs = 10 Hz.
```

```
% filter definition
plfilter = plist(); % create an empty parameters list for the filter
plfilter = append(plfilter, param('type','lowpass'));
plfilter = append(plfilter, param('win',specwin('Kaiser', 10, 150)));
plfilter = append(plfilter, param('order', 32));
plfilter = append(plfilter, param('fs', 10));
plfilter = append(plfilter, param('fc', 1));
filter = mfir(plfilter)
```

```
% resampling
pl = plist(); % create an empty parameters list
pl = append(pl, param('fsout', 1)); % add fsout = 50 Hz to parameters list
resampling process
pl = append(pl, param('filter', filter)); % use the defined filter in the
AO (y)
y = resample(x, pl); % resample the input AO (x) to obtain the resampled output
iplot(x, y) % plot original and resampled data
```



◀ Upsampling data

Interpolating data ▶

©LTP Team

Interpolating data

Interpolation of data can be done in the LTPDA Toolbox by means of [interp](#). This function interpolates the values in the input AO(s) at new values specified by the input parameter list. [Interp](#) overloads `interp1` function of Matlab Signal Processing Toolbox for AOs.

Syntaxis

```
b = interpolate(a, pl)
```

With the following parameters:

- 'vertices' – specify the new vertices to interpolate on
- 'method' – four methods are available for interpolating data
 - 'nearest' – nearest neighbor interpolation
 - 'linear' – linear interpolation
 - 'spline' – spline interpolation (default option)
 - 'cubic' – shape-preserving piecewise cubic interpolation

For details see `interp1` help of Matlab.

Examples

1. Interpolation of a sequence of random data at original sampling rate of 1 Hz by a factor of 10 with no initial phase.

```
% Signal generation
nsecs = 100;
fs     = 10;

pl = plist();
pl = append(pl, param('nsecs', nsecs));
pl = append(pl, param('fs', fs));
pl = append(pl, param('tsfcn', 'sin(2*pi*1.733*t)'));
a1 = ao(pl);

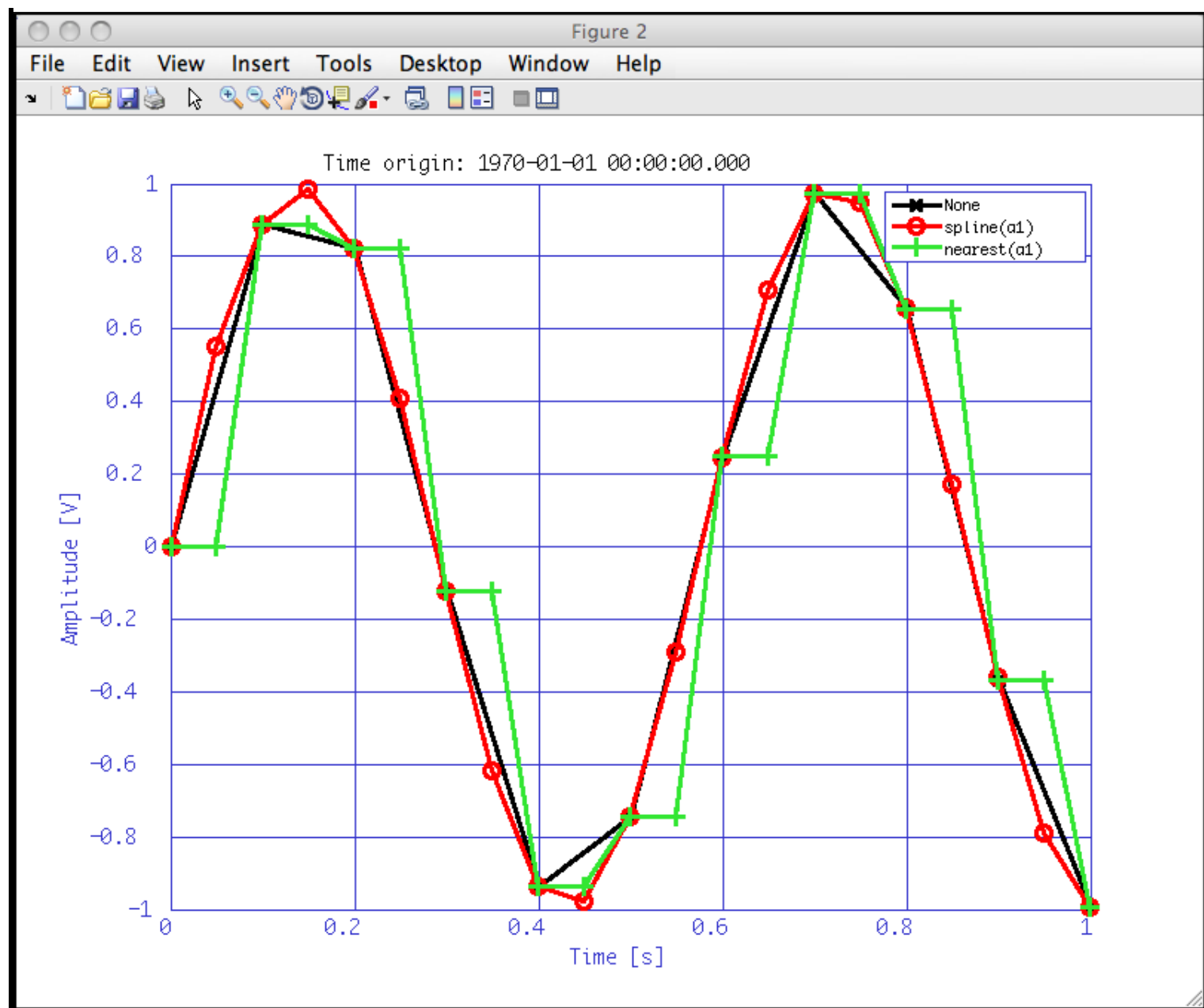
% Interpolate on a new time vector
t = linspace(0, a1.data.nsecs - 1/a1.data.fs, 2*len(a1));

plspline = plist();
plspline = append(plspline, param('vertices', t));

plnearest = plist();
plnearest = append(plnearest, param('vertices', t));
plnearest = append(plnearest, param('method', 'nearest'));

bspline = interp(a1, plspline);
bnearest = interp(a1, plnearest);

iplot([a1 bspline bnearest], plist('Markers', {'x', 'o', '+'}, 'LineColors', {'k', 'r'}));
ltlda_xaxis(0, 1)
```



◀ Resampling data

Spikes reduction in data ▶

©LTP Team

Spikes reduction in data

Spikes in data due to different nature can be removed, if desired, from the original data. LTPDA [ltpda_spikecleaning](#) detects and replaces spikes of the input AOs. A spike in data is defined as a single sample exceeding a certain value (usually, the floor noise of the data) defined by the user:

$$|x_{HPF}[n]| \leq k_{spike} \sigma_{HPF}$$

where $x_{HPF}[n]$ is the input data high-pass filtered, k_{spike} is a value defined by the user (by default is 3.3) and σ_{HPF} is the standard deviation of $x_{HPF}[n]$. In consequence, a spike is defined as the value that exceeds the floor noise of the data by a factor k_{spike} , the higher of this parameter the more difficult to "detect" a spike.

Syntaxis

```
b = ltpda_spikecleaning(a, pl)
```

With the following parameters:

- 'kspike' – set the k_{spike} value (default is 3.3)
- 'method' – method used to replace the "spiky" sample. Three methods are available ---see below for details---:
 - 'random'
 - 'mean'
 - 'previous'
- 'fc' – frequency cut-off of the high-pass IIR filter (default is 0.025)
- 'order' – specifies the order of the IIR filter (default is 2)
- 'ripple' – specifies pass/stop-band ripple for bandpass and bandreject filters (default is 0.5)

Methods explained

1. Random : this method substitutes the spiky sample by:

$$x[n] = x[n-1] + N(0, 1) \cdot \sigma_{HPF}$$

where $N(0, 1)$ is a random number of mean zero and standard deviation 1.

2. Mean : this method uses the following equation to replace the spike detected in data.

$$x[n] = \frac{x[n-1] + x[n-2]}{2}$$

3. Previous : the spike is substituted by the previous sample, i.e.:

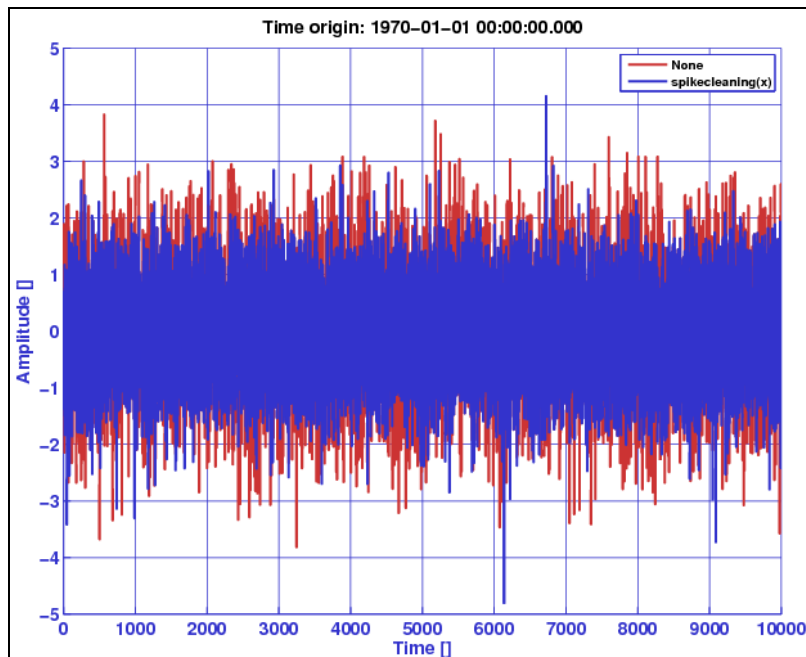
$$x[n] = x[n-1]$$

Examples

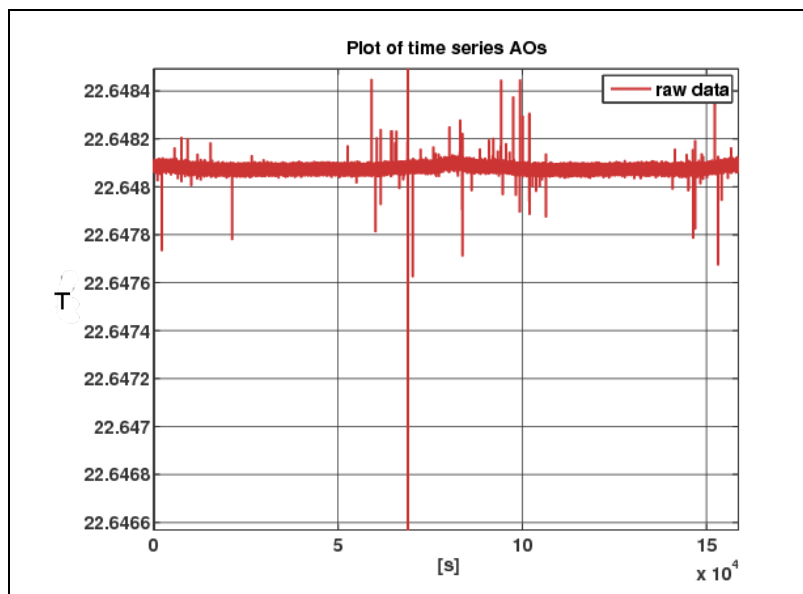
1. Spike cleaning of a sequence of random data with $k_{spike}=2$.

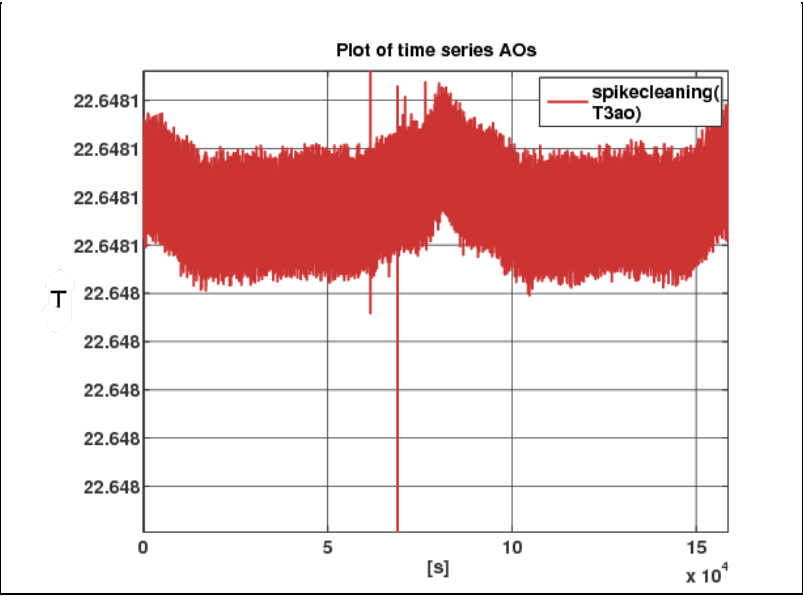
```

x = ao(tsddata(randn(10000,1),1)); % create an AO of random data sampled at 1
Hz.
pl = plist(); % create an empty parameters list
pl = append(pl, param('kspike', 2)); % kspike=2
y = ltpda_spikecleaning(x, pl); % spike cleaning function applied to the
input AO, x
iplot(x, y) % plot original and "cleaned" data
    
```



2. Example of real data: the first image shows data from the real world prior to the application of the spike cleaning tool. It is clear that some spikes are present in data and might be convenient to remove them. The second image shows the same data after the spike samples suppression.





◀ Interpolating data

Data gap filling ▶

Data gap filling

Gaps in data can be filled with *interpolated* data if desired. LTPDA [ltpda_gapfilling](#) joints two AOs by means of a segment of *synthetic* data. This segment is calculated from the two input AOs. Two different methods are possible to fill the gap in the data: *linear* and *spline*. The former fills the data by means of a linear interpolation whereas the latter uses a smoother curve ---see examples below.

Syntaxis

```
b = ltpda_gapfilling(a1, a2, pl)
```

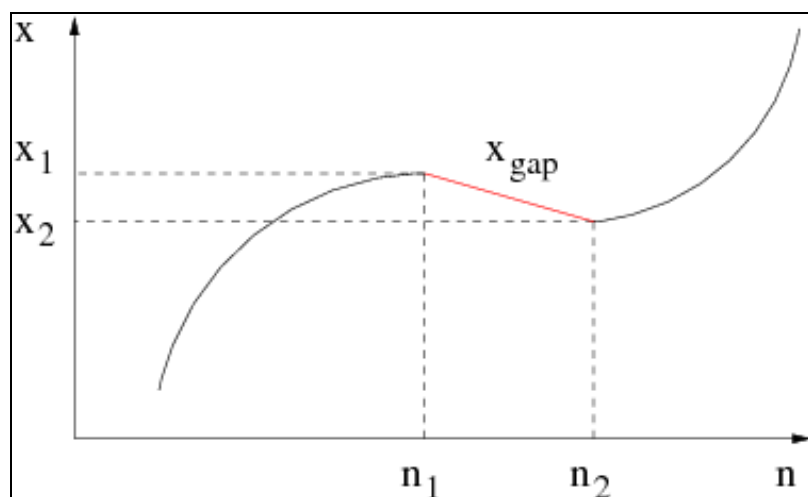
where *a1* and *a2* are the two segments to joint. The parameters are:

- 'method' - method used to interpolate missing data (see below for details)
 - 'linear' (default option)
 - 'spline'
- 'addnoise' - with this option *noise* can be added to the interpolated data. This noise is defined as a random variable with zero mean and variance equal to the high-frequency noise of the input AO.

Interpolation methods

1. Linear :

$$x_{GAP}[n] = \frac{x_2 - x_1}{n_2 - n_1}n + x + \sigma_{HPF}$$

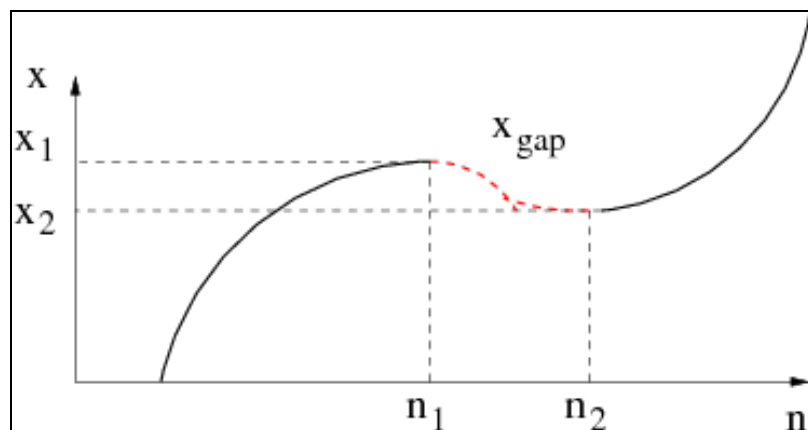


2. Spline (or third order interpolation) :

$$x_{GAP}[n] = an^3 + bn^2 + cn + d + \sigma_{HPF}$$

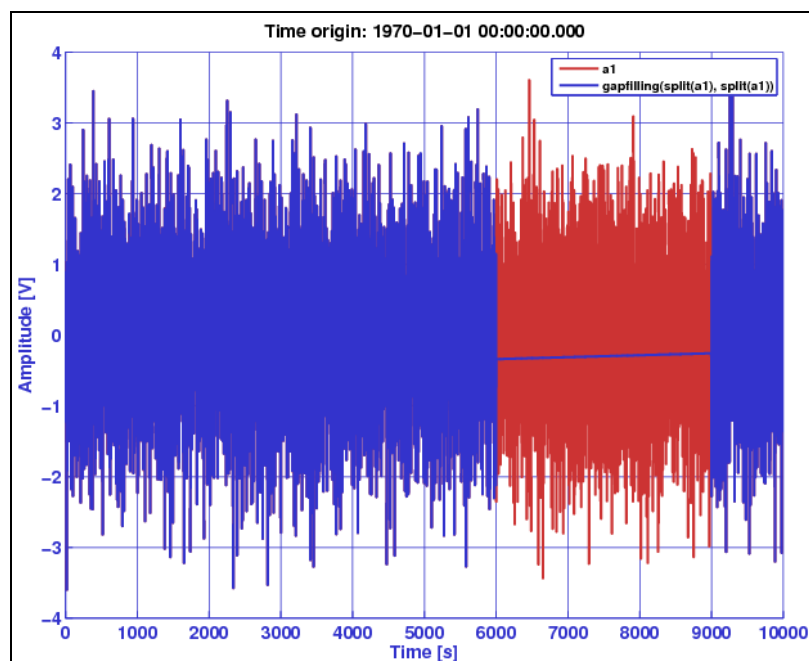
The parameters a , b , c and d are calculated by solving next system of equations:

$$\begin{aligned} x_{GAP}[n_1] &= x_1 \\ x_{GAP}[n_2] &= x_2 \\ \left. \frac{dx_{GAP}[n]}{dn} \right|_{n=n_1} &= \left. \frac{dx_1[n]}{dn} \right|_{n=n_1} \\ \left. \frac{dx_{GAP}[n]}{dn} \right|_{n=n_2} &= \left. \frac{dx_2[n]}{dn} \right|_{n=n_2} \end{aligned}$$

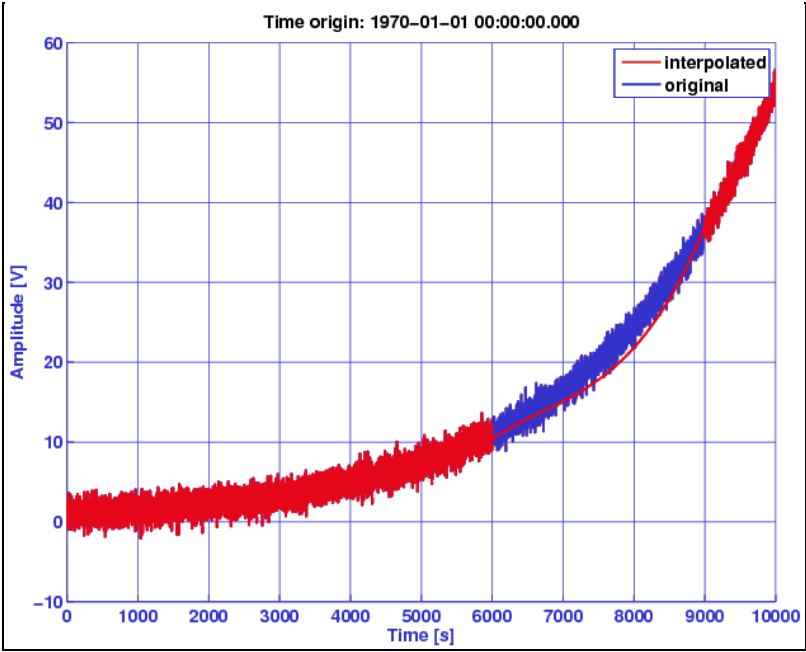


Examples

1. Missing data between two vectors of random data interpolated with the `linear` method.



2. Missing data between two data vectors interpolated with the `spline` method.



Signal Processing in LTPDA

The LTPDA Toolbox contains a set of tools to characterise digital data streams within the framework of LTPDA Objects. The current available methods can be grouped in the following categories:

- [Digital filtering](#)
- [Spectral estimation](#)
- [Fitting algorithms](#)

Digital Filtering

A digital filter is an operation that associates an input time series $x[n]$ into an output one, $y[n]$. Methods developed in the LTPDA Toolbox deal with linear digital filters, i.e. those which fulfill that a linear combination of inputs results in a linear combination of outputs with the same coefficients (provided that these are not time dependent). In these conditions, the filter can be expressed as

$$y[n] = \sum_{k=-\infty}^{\infty} h[k] x[n-k]$$

described in these terms, the filter is completely described by the impulse response $h[k]$, and can then be subdivided into the following classes:

- Causal: if there is no output before input is fed in.

$$h[k] = 0, k < 0$$

- Stable: if finite input results in finite output.

$$\sum_{k=-\infty}^{\infty} h[k] < \infty$$

- Shift invariant: if time shift in the input results in a time shift in the output by the same amount.

$$h[k] = h[-k]$$

Digital filters classification

Digital filters can be described as difference equations. If we consider an input time series x and an output y , three specific cases can then be distinguished:

- Autoregressive (AR) process: the difference equation in this case is given by:

$$y[n] = \sum_{k=1}^M b[k] y[n-k]$$

AR processes can be also classified as [IIR Filters](#).

- Moving Average (MA) process: the difference equation in this case is given by:

$$y[n] = \sum_{k=0}^N a[k] x[n-k]$$

MA processes can be also classified as [FIR Filters](#).

- Autoregressive Moving Average (ARMA) process: the difference equation in this case contains both an AR and a MA process:

$$y[n] = \sum_{k=0}^N b[k] x[n-k] - \sum_{k=1}^M a[k] y[n-k]$$

◀ Signal Processing in LTPDA

IIR Filters ▶

©LTP Team

IIR Filters

Infinite Impulse Response filters are those filters present a non-zero infinite length response when excited with a very brief (ideally an infinite peak) input signal. A linear causal IIR filter can be described by the following difference equation

$$y[n] = \sum_{k=0}^N b[k] x[n-k] - \sum_{k=1}^M a[k] y[n-k]$$

This operation describe a recursive system, i.e. a system that depends on current and past samples of the input $x[n]$, but also on the output data stream $y[n]$.

Creating a IIR filter in the LTPDA

The LTPDA Toolbox allows the implementation of IIR filters by means of the [miir class](#).

Creating from a plist

The following example creates an order 1 highpass filter with high frequency gain 2. Filter is designed for 10 Hz sampled data and has a cut-off frequency of 0.2 Hz.

```
>> pl = plist('type', 'highpass', ...
              'order', 1,          ...
              'gain', 2.0,         ...
              'fs', 10,           ...
              'fc', 0.2);
>> f = miir(pl)
```

Creating from a pzmodel

IIR filters can also be [created from a pzmodel](#).

Creating from a difference equation

Alternatively, the filter can be defined in terms of two vectors specifying the coefficients of the filter and the sampling frequency. The following example creates a IIR filter with sampling frequency 1 Hz and the following recursive equation:

$$y[n] = 0.5x[n] - 0.01x[n-1] - 0.1y[n-1]$$

```
>> a = [0.5 -0.01];
>> b = [1 0.1];
>> fs = 1;
>> f = miir(a,b,fs)
```

Notice that the convention used in this function is the one described in the [Digital filters classification](#) section

Importing an existing model

The miir constructor also accepts as an input existing models in different formats:

-
- LISO files:

```
>> f = miir('foo_iir.fil')
```

- XML files:

```
>> f = miir('foo_iir.xml')
```

- MAT files:

```
>> f = miir('foo_iir.mat')
```

- From repository:

```
>> f = miir(plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []))
```

[◀ Digital Filtering](#)[FIR Filters ▶](#)

©LTP Team

FIR Filters

Finite Impulse Response filters are those filters present a non-zero finite length response when excited with a very brief (ideally an infinite peak) input signal. A linear causal FIR filter can be described by the following difference equation

$$y[n] = \sum_{k=0}^M b[k] x[n-k]$$

This operation describe a nonrecursive system, i.e. a system that only depends on current and past samples of the input data stream $x[n]$

Creating a FIR filter in the LTPDA

The LTPDA Toolbox allows the implementation of FIR filters by means of the [mfir class](#).

Creating from a plist

The following example creates an order 64 highpass filter with high frequency gain 2. Filter is designed for 1 Hz sampled data and has a cut-off frequency of 0.2 Hz.

```
>> pl = plist('type', 'highpass', ...
             'order', 64, ...
             'gain', 2.0, ...
             'fs', 1, ...
             'fc', 0.2);
>> f = mfir(pl)
```

Creating from a difference equation

The filter can be defined in terms of two vectors specifying the coefficients of the filter and the sampling frequency. The following example creates a FIR filter with sampling frequency 1 Hz and the following recursive equation:

$$y[n] = -0.8x[n] + 10x[n-1]$$

```
>> b = [-0.8 10];
>> fs = 1;
>> f = mfir(b, fs)
```

Creating from an Analysis Object

A FIR filter can be generated based on the magnitude of the input Analysis Object or fsdata object. In the following example a fsdata object is first generated and then passed to the mfir constructor to obtain the equivalent FIR filter.

```
>> fs = 10;                $ sampling frequency
>> f = linspace(0, fs/2, 1000);
>> y = 1./(1+(0.1^2*pi*f).^2); $ an arbitrary function
```

```
>> fsd = fsdata(f,y,fs);           $ build the fsdata object
>> f = mfir(ao(fsd));
```

Available methods for this option are: 'frequency-sampling' (uses fir2), 'least-squares' (uses firls) and 'Parks-McClellan' (uses firpm)

Importing an existing model

The mfir constructor also accepts as an input existing models in different formats:

-
- LISO files:

```
>> f = mfir('foo_fir.fil')
```

- XML files:

```
>> f = mfir('foo_fir.xml')
```

- MAT files:

```
>> f = mfir('foo_fir.mat')
```

- From repository:

```
>> f = mfir(plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []))
```

◀ IIR Filters

Spectral Estimation ▶

©LTP Team

Spectral Estimation

Spectral estimation is a branch of the signal processing, performed on data and based on frequency-domain techniques. Within the LTPDA toolbox many functions of the Matlab Signal Processing Toolbox (which is required) were rewritten to operate on LTPDA Analysis Objects. Univariate and multivariate technique are available, so to estimate for example the linear power spectral density or the cross spectral density of one or more signals. The focus of the tools is on time-series objects, whose spectral content needs to be estimated.

The power spectrum density estimators are based on pwelch from MATLAB, which is an implementation of Welch's averaged, modified periodogram method.

The following pages describe the different Welch-based spectral estimation tools available in the LTPDA toolbox:

- [LTPDA_PWELCH power spectral density estimates](#)
- [LTPDA_CPSD cross-spectral density estimates](#)
- [LTPDA_COHERENCE magnitude squared coherence estimates](#)
- [LTPDA_TFE transfer function estimates](#)

As an alternative, the LTPDA toolbox makes available the same set of estimators, based on an implementation of the LPSD algorithm (See "Improved spectrum estimation from digitized time series on a logarithmic frequency axis", M Troebs, G Heinzl, [Measurement 39 \(2006\) 120–129](#)).

The following pages describe the different LPSD-based spectral estimation tools available in the LTPDA toolbox:

- [LTPDA_LPSDLog-scale power spectral density estimates](#)
- [LTPDA_LCPSDLog-scale cross-spectral density estimates](#)
- [LTPDA_LCOHERENCELog-scale magnitude squared coherence estimates](#)
- [LTPDA_LTFELog-scale transfer function estimates](#)

More detailed help on spectral estimation can be found in the help associated with the Signal Processing Toolbox (>> doc signal)

Power spectral density estimates

Univariate power spectral density is performed by the Welch's averaged, modified periodogram method. [ltpda_pwelch](#) estimates the power spectral density of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectrum, by multiplying it with a [spectral window object](#), and can be detrended by polynomial of time in order to reduce the impact of the border discontinuities. The window length is adjustable to shorter lengths to reduce the spectral density uncertainties, and the percentage of subsequent window overlap can be adjusted as well. The detrending is performed on the individual windows. The user can choose the quantity being given in output among ASD (amplitude spectral density), PSD (power spectral density), AS (amplitude spectrum), and PS (power spectrum).

Syntaxis

```
b = ltpda_pwelch(a1,a2,a3,...,pl)
```

a1, *a2*, *a3*, ... are AO(s) containing the input time series to be evaluated. *b* includes the output object(s). The parameter list *pl* includes the following parameters:

- 'win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'Nfft' – number of samples in each fft [default: length of input data]
- 'Scale' – one of
 - 'ASD' – amplitude spectral density
 - 'PSD' – power spectral density [default]
 - 'AS' – amplitude spectrum
 - 'PS' – power spectrum
- 'order' – order of segment detrending
 - -1 – no detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

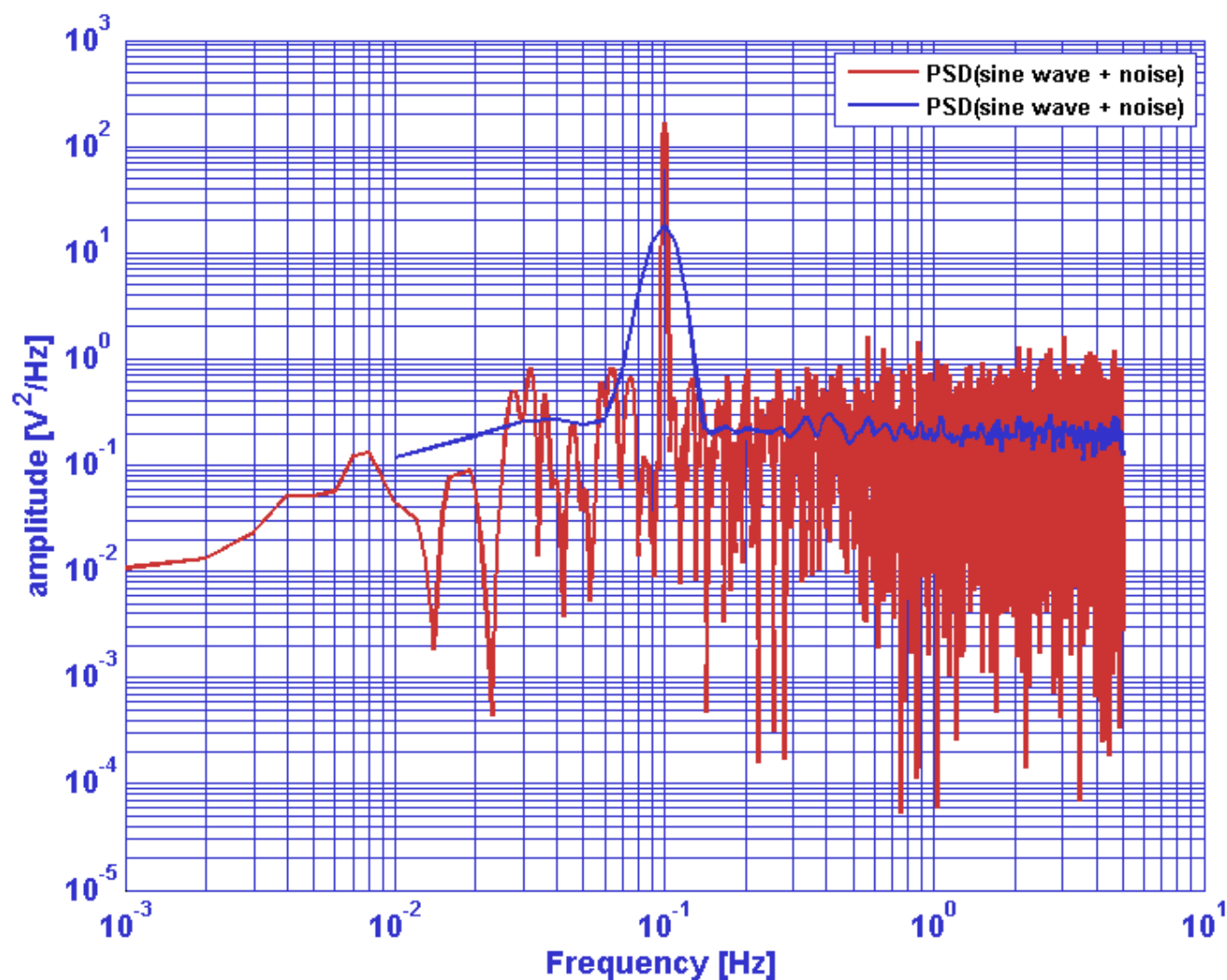
The length of the window is set by the value of the parameter 'Nfft', so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

If the user doesn't specify the value of a given parameter, the default value is used.

Examples

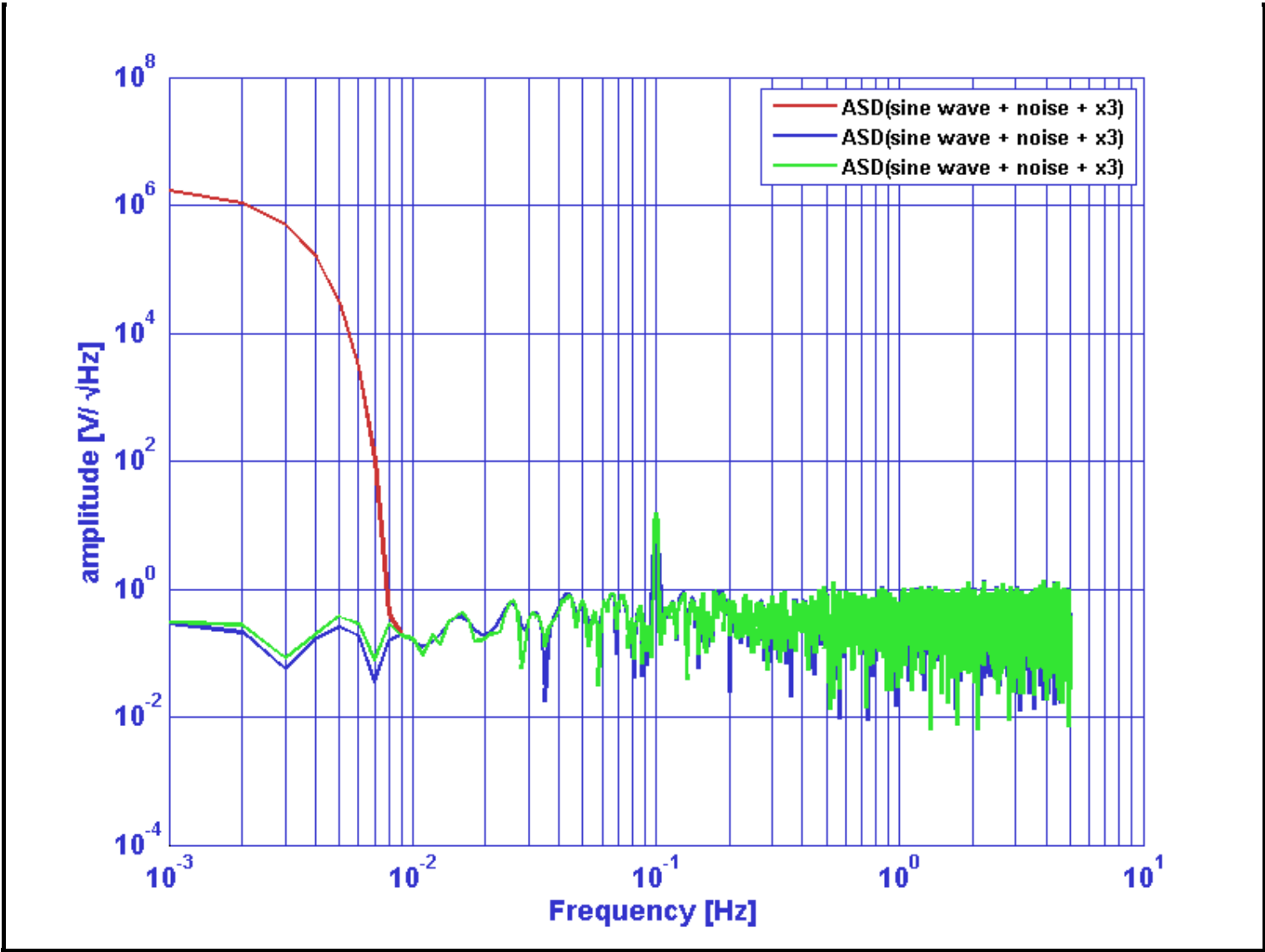
1. Evaluation of the PSD of a time-series represented by a low frequency sinewave signal, superimposed to white noise. Comparison of the effect of windowing on the estimate of the white noise level and on resolving the signal.

```
x1 = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10));
x2 = ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
x = x1 + x2;
y_lf = ltpda_pwelch(x);
y_hf = ltpda_pwelch(x,plist('nfft',1000));
ipplot(y_lf, y_hf)
```



2. Evaluation of the PSD of a time-series represented by a low frequency sinewave signal, superimposed to white noise and to a low frequency linear drift. Comparison of the effect of different windows and of detrending.

```
x1 = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10));
x2 = ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
x3 = ao(plist('tsfcn','t.^2 + t','nsecs',1000,'fs',10));
x = x1 + x2 + x3;
y_1 = ltpda_pwelch(x,plist('scale','ASD','order',1));
y_2 = ltpda_pwelch(x,plist('scale','ASD','order',2));
y_3 = ltpda_pwelch(x,plist('scale','ASD','order',2,'win',specwin(plist('name','BH92'))));
iplot(y_1, y_2, y_3);
```



◀ Spectral Estimation

Cross-spectral density estimates ▶

©LTP Team

Cross-spectral density estimates

Multivariate power spectral density is performed by the Welch's averaged, modified periodogram method. [ltpda_cpsd](#) estimates the cross-spectral density of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectra, by multiplying it with a [spectral window object](#), and can be detrended by polynomial of time in order to reduce the impact of the border discontinuities. The window length is adjustable to shorter lengths to reduce the spectral density uncertainties, and the percentage of subsequent window overlap can be adjusted as well.

Syntaxis

```
b = ltpda_cpsd(a1,a2,a3,...,pl)
```

a1, a2, a3, ... are AOs containing the input time series to be evaluated. They need to be in a number N >= 2. b includes the NXN output objects. The parameter list pl includes the following parameters:

- 'win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'Nfft' – number of samples in each fft [default: length of input data]
- 'order' – order of segment detrending
 - -1 – no detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

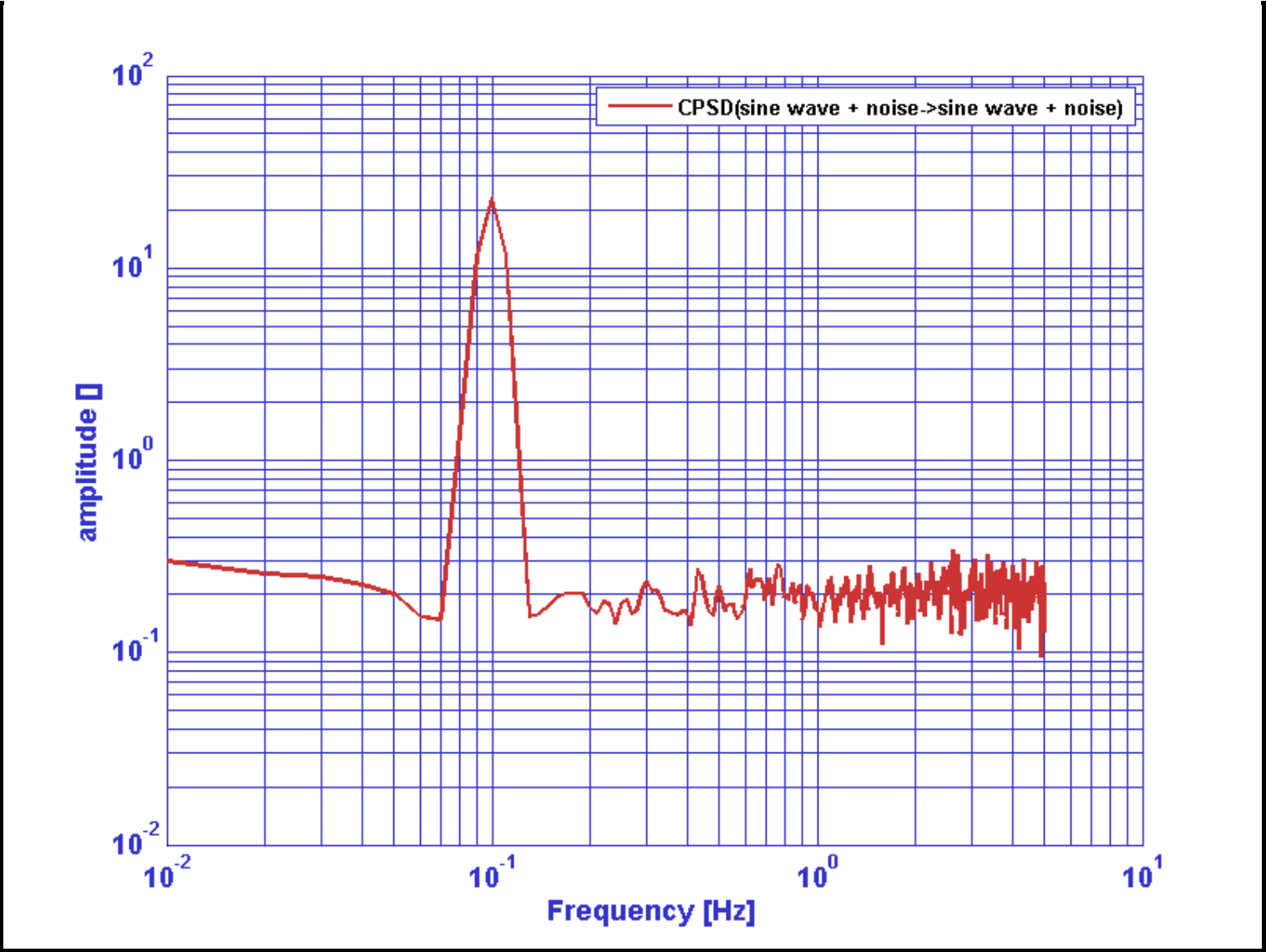
The length of the window is set by the value of the parameter 'Nfft', so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

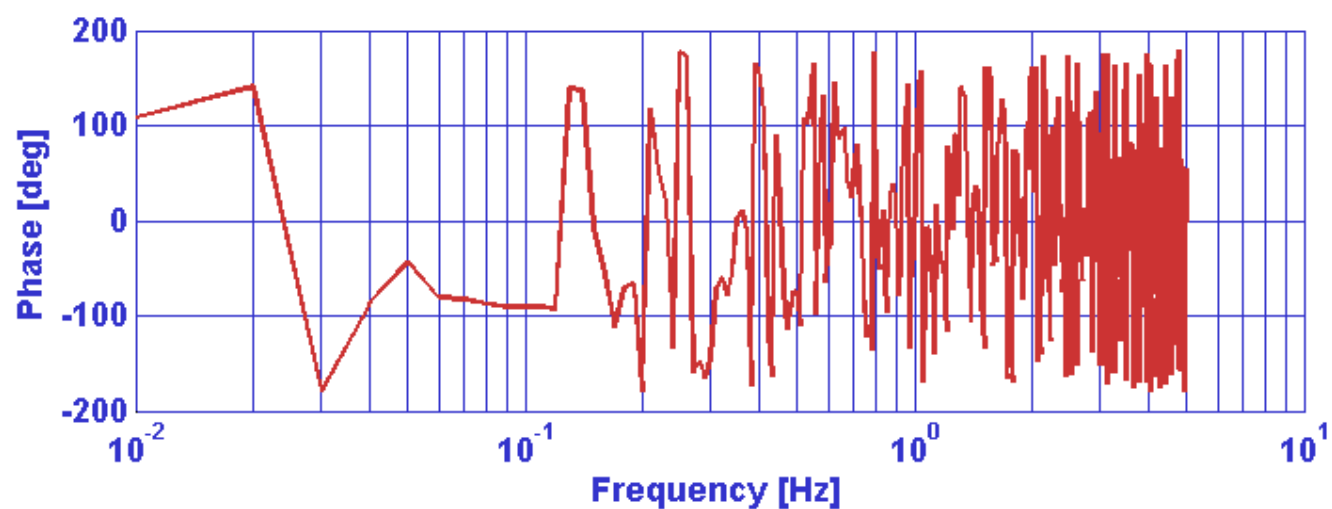
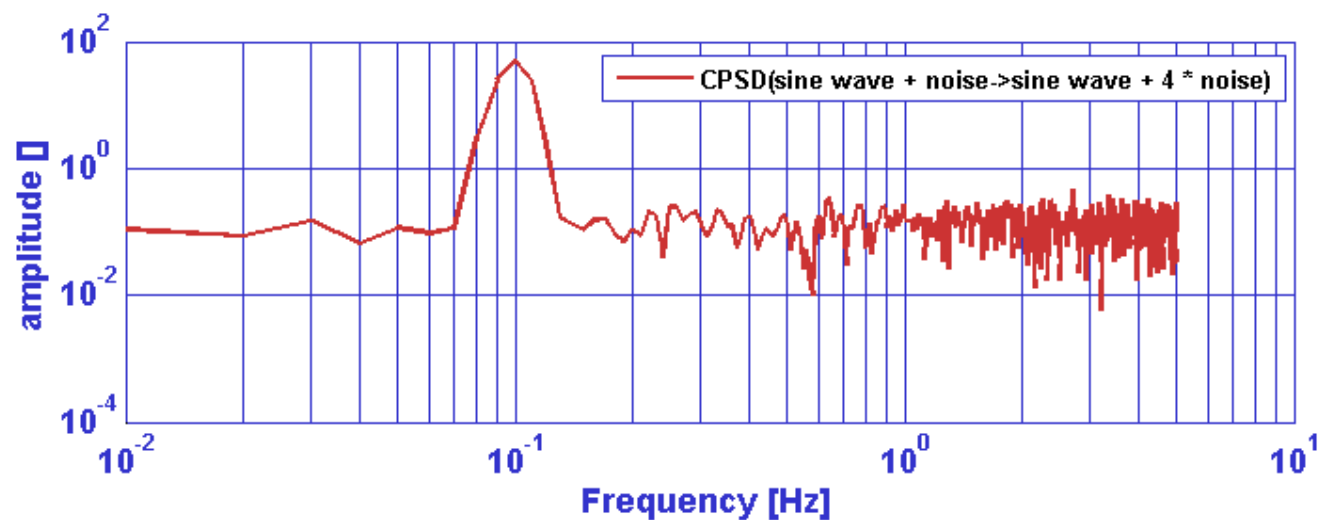
If the user doesn't specify the value of a given parameter, the default value is used. The function makes CPSD estimates between all input AOs. Therefore, if the input argument list contains N analysis objects, the output a will contain NxN CPSD estimates. The diagonal elements will be S_{ai_ai} and will be equivalent to the output of ltpda_pwelch(ai).

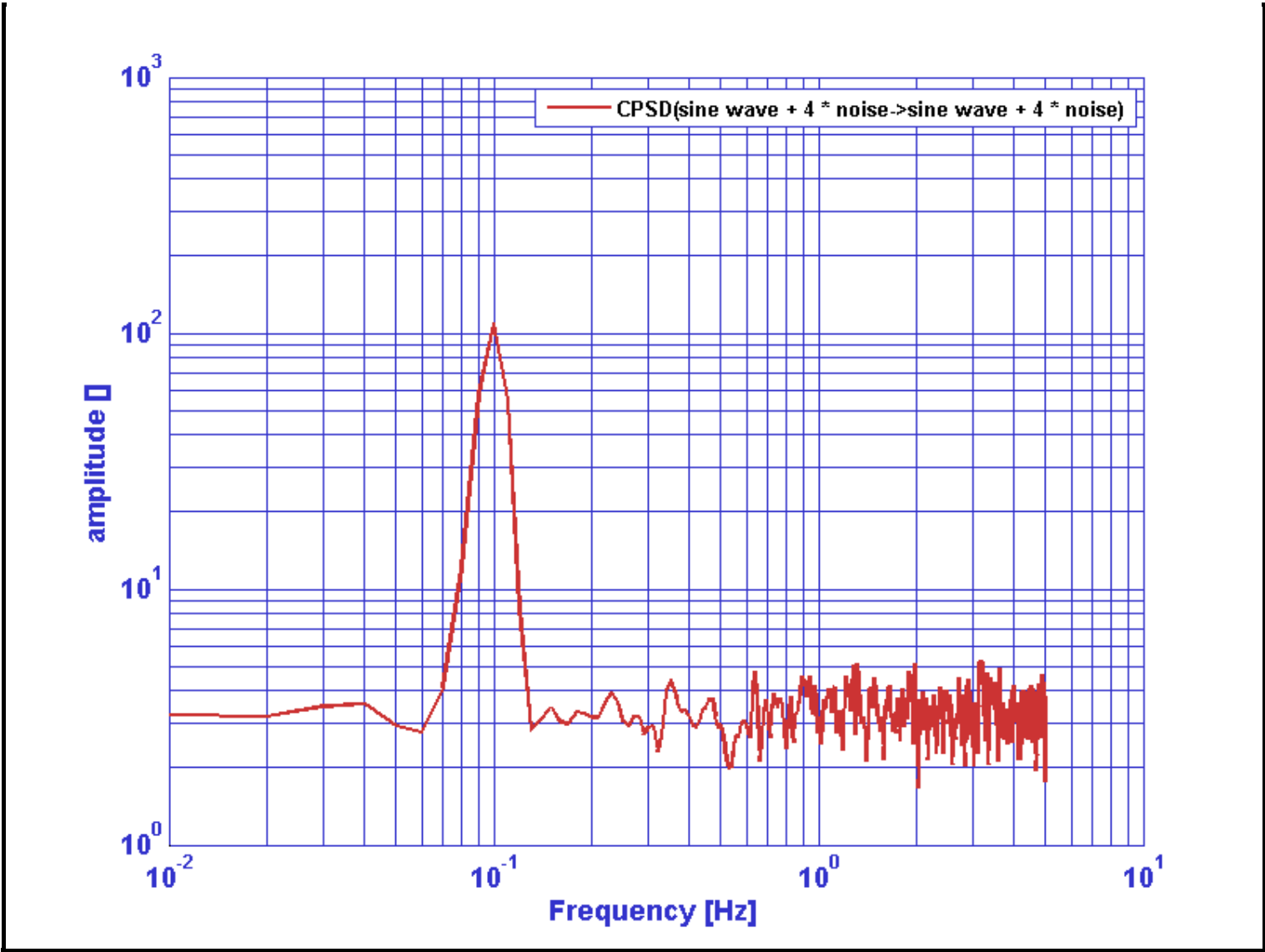
Example

Evaluation of the CPSD of two time-series represented by: a low frequency sinewave signal superimposed to white noise, and a low frequency sinewave signal at the same frequency, phase shifted and with different amplitude, superimposed to white noise.

```
x = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10)) + ...
ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
y = ao(plist('waveform','sine wave','f',0.1,'A',2,'nsecs',1000,'fs',10,'phi',90)) + ...
4*ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
z = ltpda_cpsd(x,y,plist('nfft',1000));
iplot(z(1,1));
iplot(z(1,2));
iplot(z(2,2));
```





Cross coherence estimates

Multivariate power spectral density is performed by the Welch's averaged, modified periodogram method. [ltpda_cohere](#) estimates the coherence of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectra, by multiplying it with a [spectral window object](#), and can be detrended by polinomial of time in order to reduce the impact of the border discontinuities. The window length is adjustable to shorter lenghts to reduce the spectral density uncertainties, and the percentage of subsequent window overlap can be adjusted as well.

Syntaxis

```
b = ltpda_cohere(a1,a2,a3,...,pl)
```

a1, a2, a3, ... are AOs containing the input time series to be evaluated. They need to be in a number N >= 2. b includes the NXN output objects. The parameter list pl includes the following parameters:

- 'win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'Nfft' – number of samples in each fft [default: length of input data]
- 'order' – order of segment detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

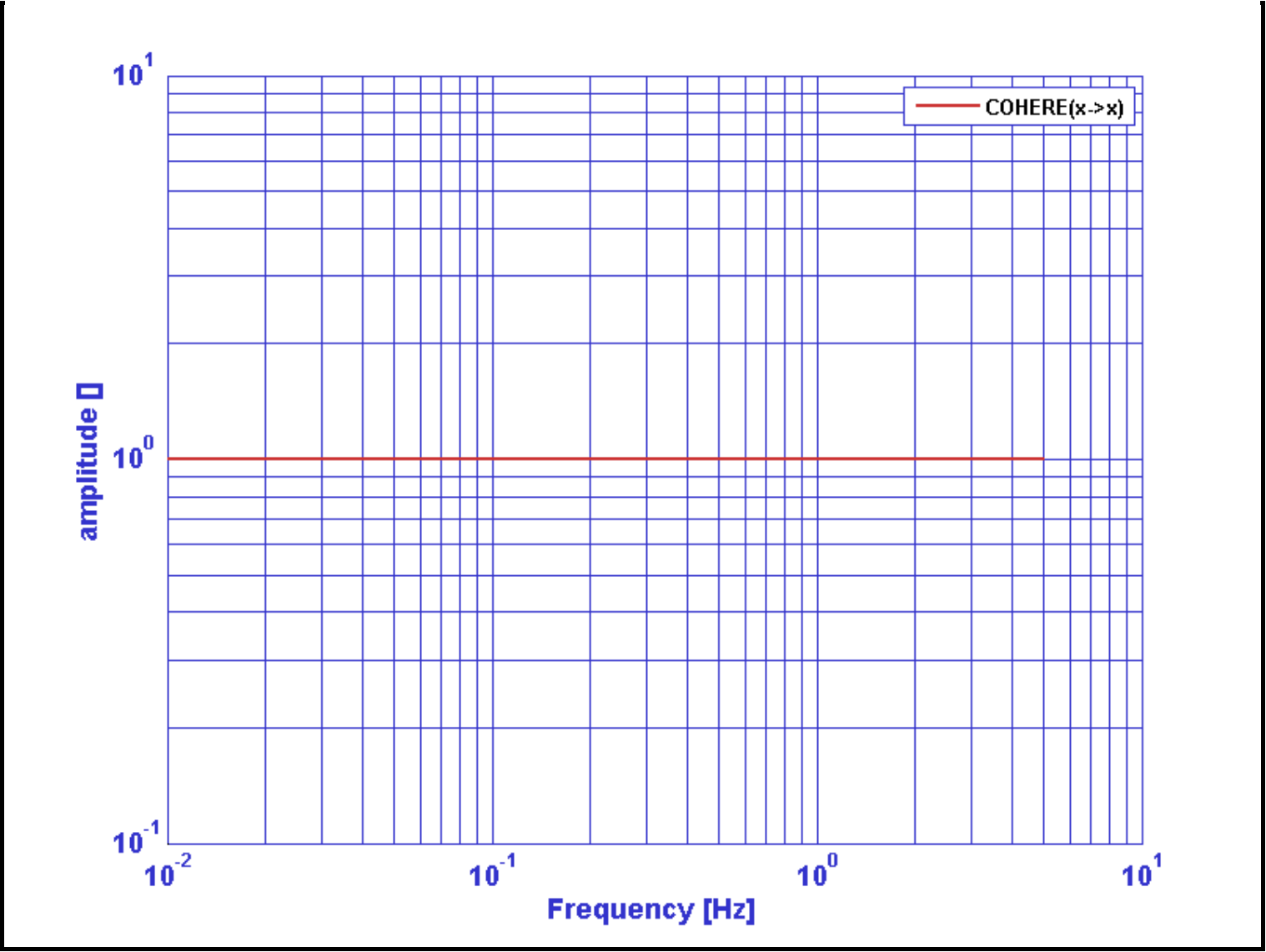
The length of the window is set by the value of the parameter 'Nfft', so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

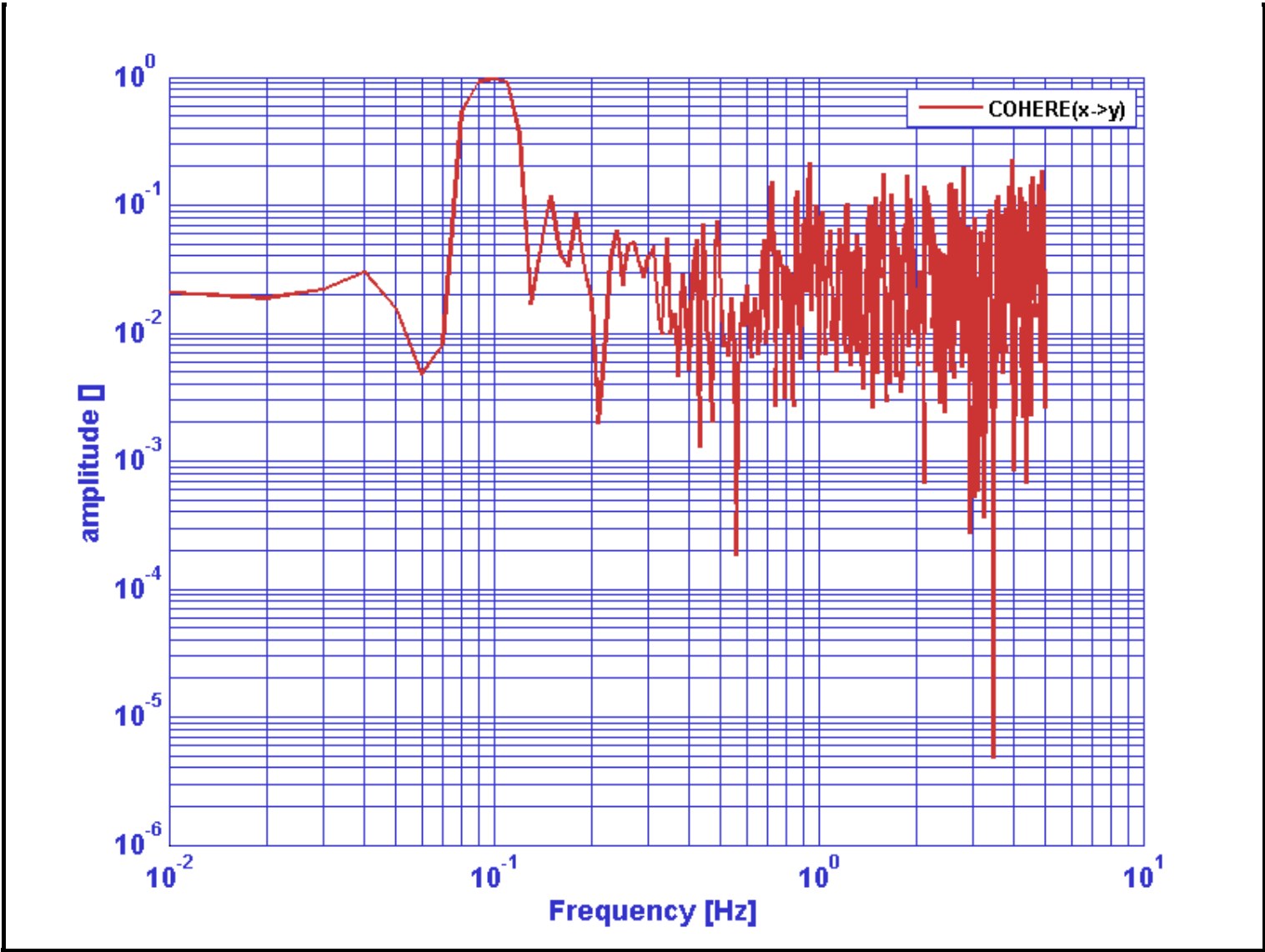
If the user doesn't specify the value of a given parameter, the default value is used. The function makes coherence estimates between all input AOs. Therefore, if the input argument list contains N analysis objects, the output a will contain NxN coherence estimates. The diagonal elements will be 1.

Example

Evaluation of the coherence of two time-series represented by: a low frequency sinewave signal superimposed to white noise, and a low frequency sinewave signal at the same frequency, phase shifted and with different amplitude, superimposed to white noise.

```
x = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10)) + ...
ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10)) + ...
ao(plist('tsfcn','t','nsecs',1000,'fs',10));
y = ao(plist('waveform','sine wave','f',0.1,'A',2,'nsecs',1000,'fs',10,'phi',90)) + ...
4*ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
pl = plist('win',specwin(plist('name','BH92')),'nfft',1000);
z = ltpda_cohere(x,y,pl);
iplot(z(1,1));
iplot(z(1,2));
```





Transfer function estimates

Multivariate power spectral density is performed by the Welch's averaged, modified periodogram method. [ltpda_tfe](#) estimates the transfer function of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectra, by multiplying it with a [spectral window object](#), and can be detrended by polynomial of time in order to reduce the impact of the border discontinuities. The window length is adjustable to shorter lengths to reduce the spectral density uncertainties, and the percentage of subsequent window overlap can be adjusted as well.

Syntaxis

```
b = ltpda_tfe(a1,a2,a3,...,pl)
```

a1, a2, a3, ... are AOs containing the input time series to be evaluated. They need to be in a number N >= 2. b includes the NXN output objects. The parameter list pl includes the following parameters:

- 'win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'nfft' – number of samples in each fft [default: length of input data / 4]
- 'order' – order of segment detrending
 - -1 – no detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

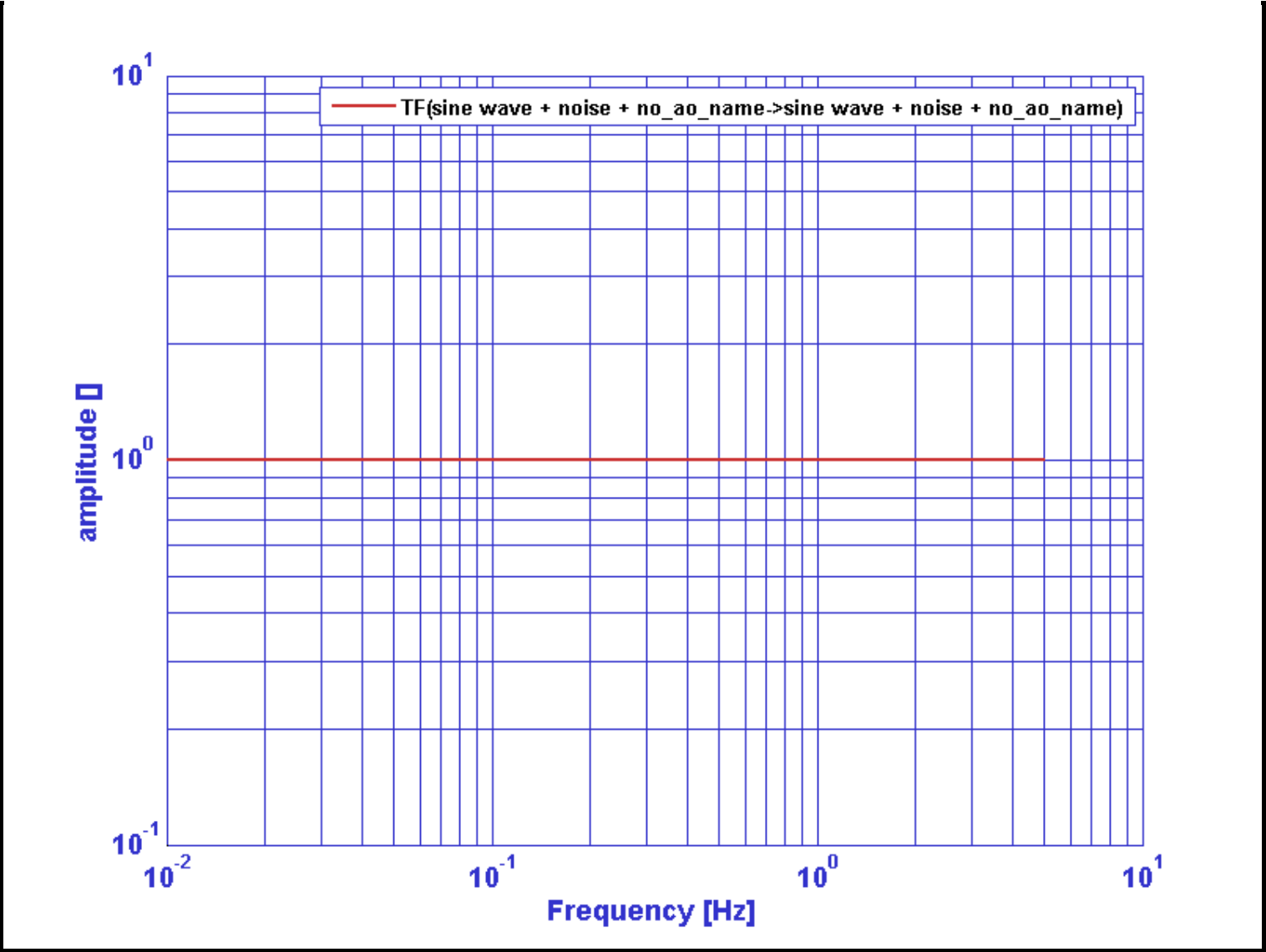
The length of the window is set by the value of the parameter 'nfft', so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLI.

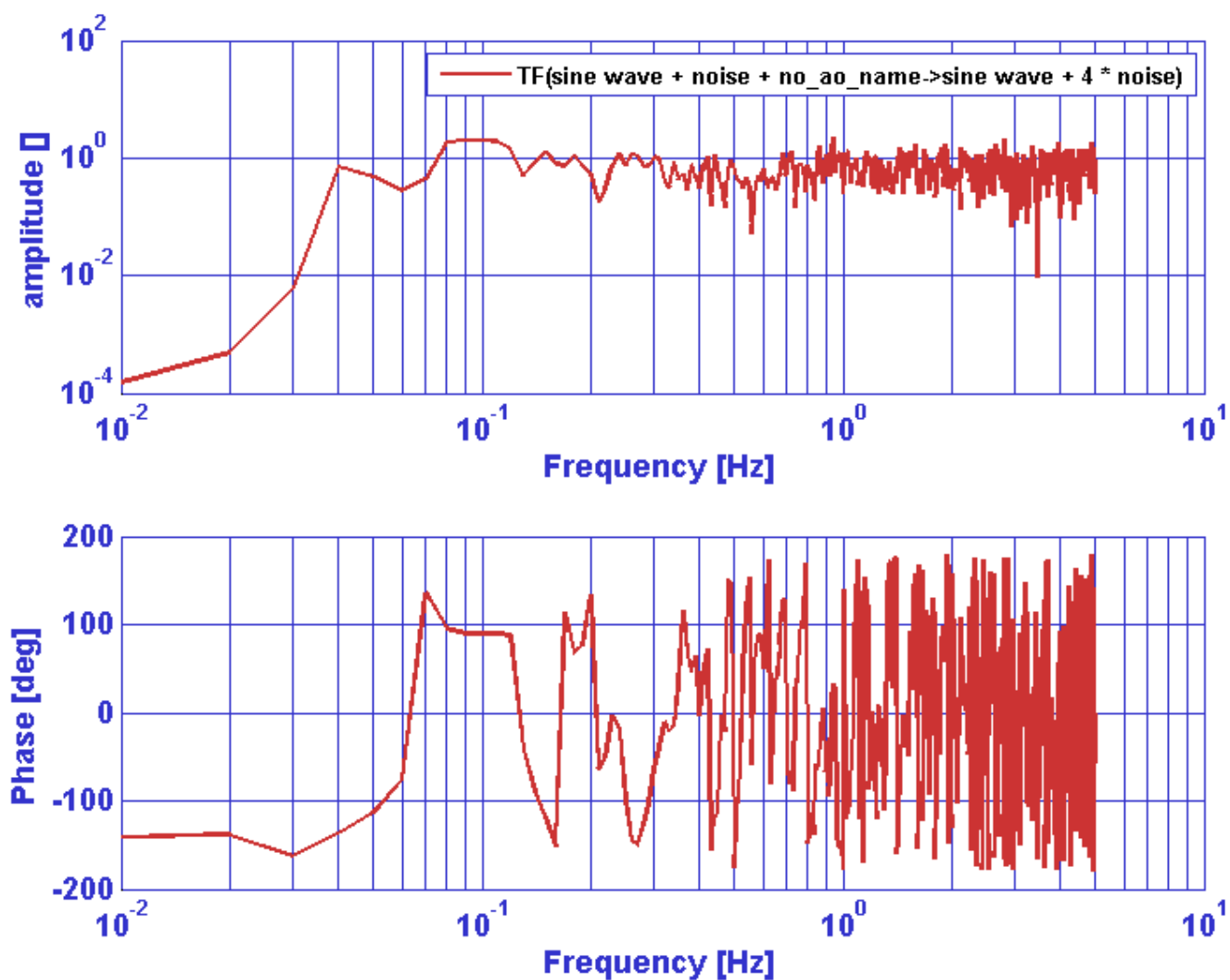
If the user doesn't specify the value of a given parameter, the default value is used. The function makes transfer functions estimates between all input AOs. Therefore, if the input argument list contains N analysis objects, the output a will contain NxN TFE estimates. The diagonal elements will be 1.

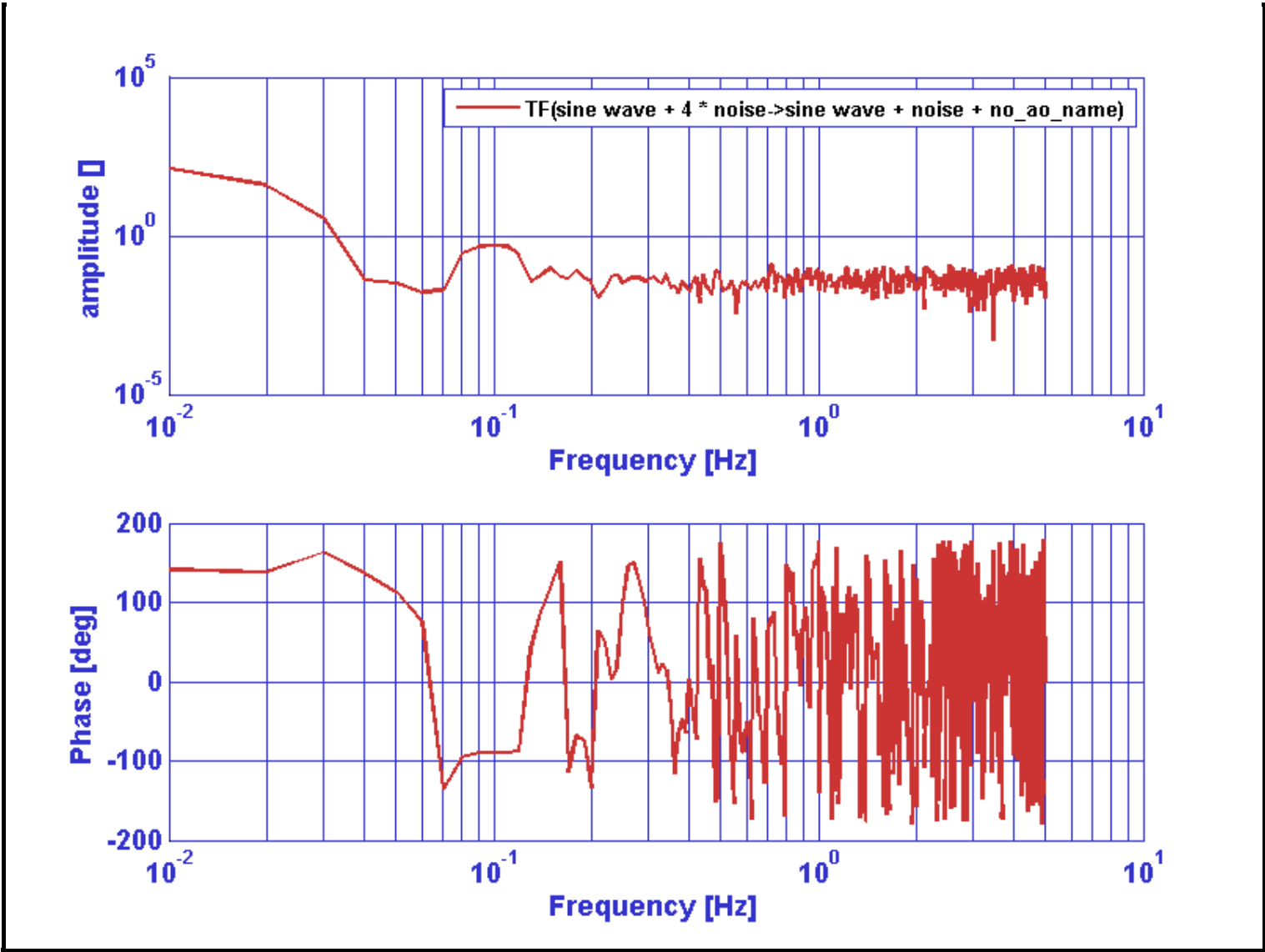
Example

Evaluation of the transfer function between two time-series represented by: a low frequency sinewave signal superimposed to white noise, and a low frequency sinewave signal at the same frequency, phase shifted and with different amplitude, superimposed to white noise.

```
x = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10)) + ...
ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
y = ao(plist('waveform','sine wave','f',0.1,'A',2,'nsecs',1000,'fs',10,'phi',90)) + ...
4*ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
z = ltpda_tfe(x,y,plist('win',specwin(plist('name','BH92')),'nfft',1000));
iplot(z(1,1));
iplot(z(1,2));
iplot(z(2,1));
```







Log-scale power spectral density estimates

Univariate power spectral density on a logarithmic scale can be performed by the LPSD algorithm, which is an application of Welch's averaged, modified periodogram method, spectral density estimates are not evaluated at frequencies which are linear multiples of the minimum frequency resolution $1/\tau$ where τ is the window length, but on a logarithmic scale. The algorithm takes care of calculating the frequencies at which to evaluate the spectral estimate, aiming at minimizing the uncertainty in the estimate itself, and to recalculate a suitable window length for each frequency bin.

[ltpda_lpsd](#) estimates the power spectral density of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectrum, by multiplying it with a [spectral window object](#), and can be detrended by polynomial of time in order to reduce the impact of the border discontinuities. Detrending is performed on each individual window. The user can choose the quantity being given in output among ASD (amplitude spectral density), PSD (power spectral density), AS (amplitude spectrum), and PS (power spectrum).

Syntaxis

```
b = ltpda_lpsd(a1,a2,a3,...,pl)
```

$a1$, $a2$, $a3$, ... are AO(s) containing the input time series to be evaluated. b includes the output object(s). The parameter list pl includes the following parameters:

- 'kdes' – the desired number of averages [default: 100]
- 'jdes' – the number of spectral frequencies to compute [default: $fs/2$]
- 'Lmin' – the minimum segment length [default: 0]
- 'win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'Scale' – one of
 - 'ASD' – amplitude spectral density
 - 'PSD' – power spectral density [default]
 - 'AS' – amplitude spectrum
 - 'PS' – power spectrum
- 'order' – order of segment detrending
 - -1 – no detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

The length of the window is recalculated for each frequency bin, so the 'Win' parameter is used only to provide the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

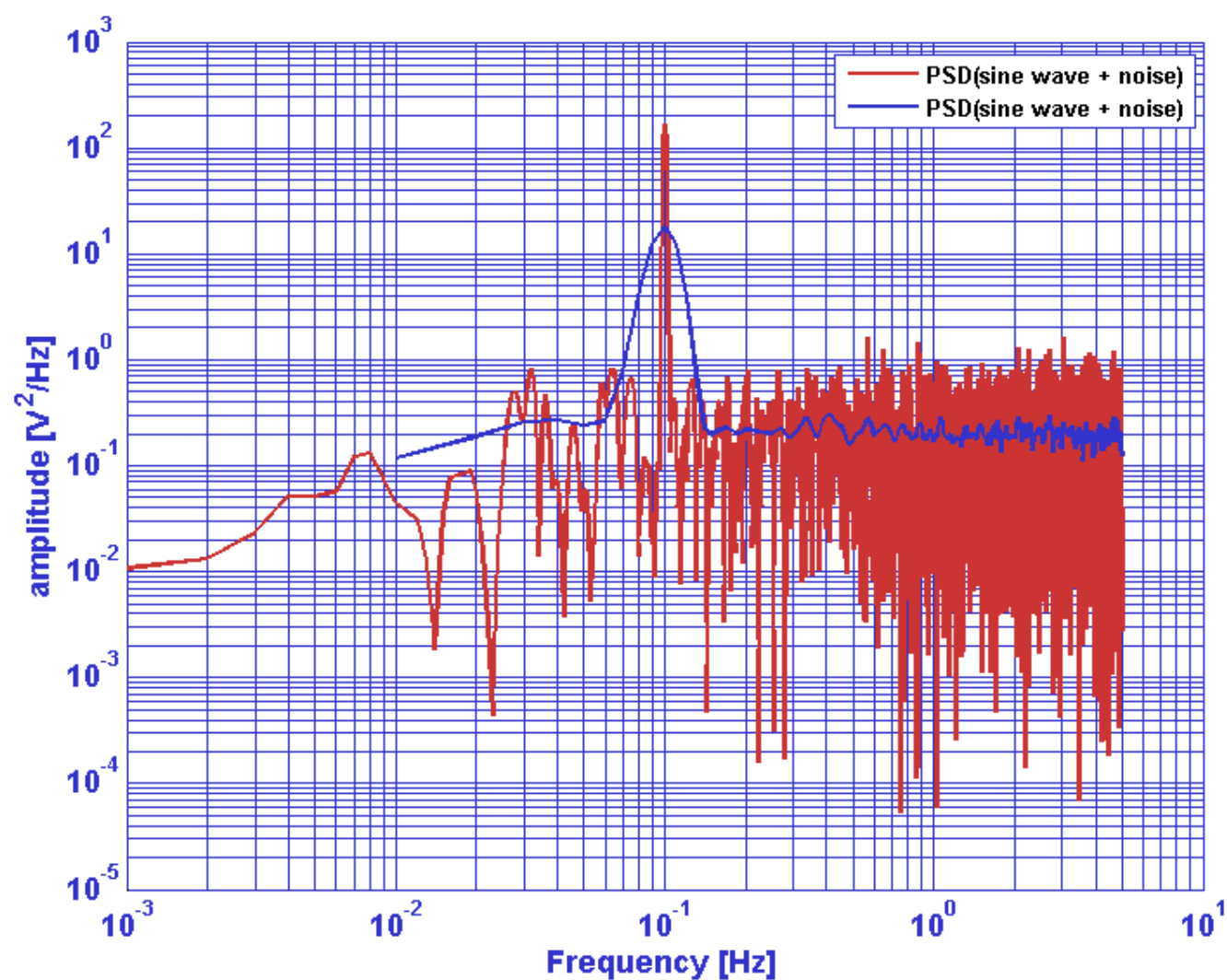
If the user doesn't specify the value of a given parameter, the default value is used.

Examples

1. Evaluation of the ASD of a time-series represented by a low frequency sinewave signal, superimposed to white noise. Comparison of the effect of using standard Pwelch and LPSD on the estimate of the white noise level and on resolving the signal.

```
x1 = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10));
x2 = ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
```

```
x = x1 + x2;
pl = plist('scale','ASD','order',-1,'win',specwin(plist('name','BH92')));
y1 = ltpda_pwelch(x, pl);
y2 = ltpda_lpsd(x, pl);
ipplot(y1, y2)
```



◀ Transfer function estimates

Log-scale cross-spectral density estimates ▶

©LTP Team

Log-scale cross-spectral density estimates

Multivariate power spectral density on a logarithmic scale can be performed by the LPSD algorithm, which is an application of Welch's averaged, modified periodogram method, cross-spectral density estimates are not evaluated at frequencies which are linear multiples of the minimum frequency resolution $1/T$ where T is the window length, but on a logarithmic scale. The algorithm takes care of calculating the frequencies at which to evaluate the spectral estimate, aiming at minimizing the uncertainty in the estimate itself, and to recalculate a suitable window length for each frequency bin.

[ltpda_lcpsd](#) estimates the cross-spectral density of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectra, by multiplying it with a [spectral window object](#), and can be detrended by polynomial of time in order to reduce the impact of the border discontinuities. Detrending is performed on each individual window.

Syntaxis

```
b = ltpda_lcpsd(a1,a2,a3,...,pl)
```

$a1, a2, a3, \dots$ are AOs containing the input time series to be evaluated. They need to be in a number $N \geq 2$. b includes the $N \times N$ output objects. The parameter list pl includes the following parameters:

- 'Kdes' – the desired number of averages [default: 100]
- 'Jdes' – the number of spectral frequencies to compute [default: $f_s/4$]
- 'Lmin' – the minimum segment length [default: 0]
- 'Win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'order' – order of segment detrending
 - -1 – no detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

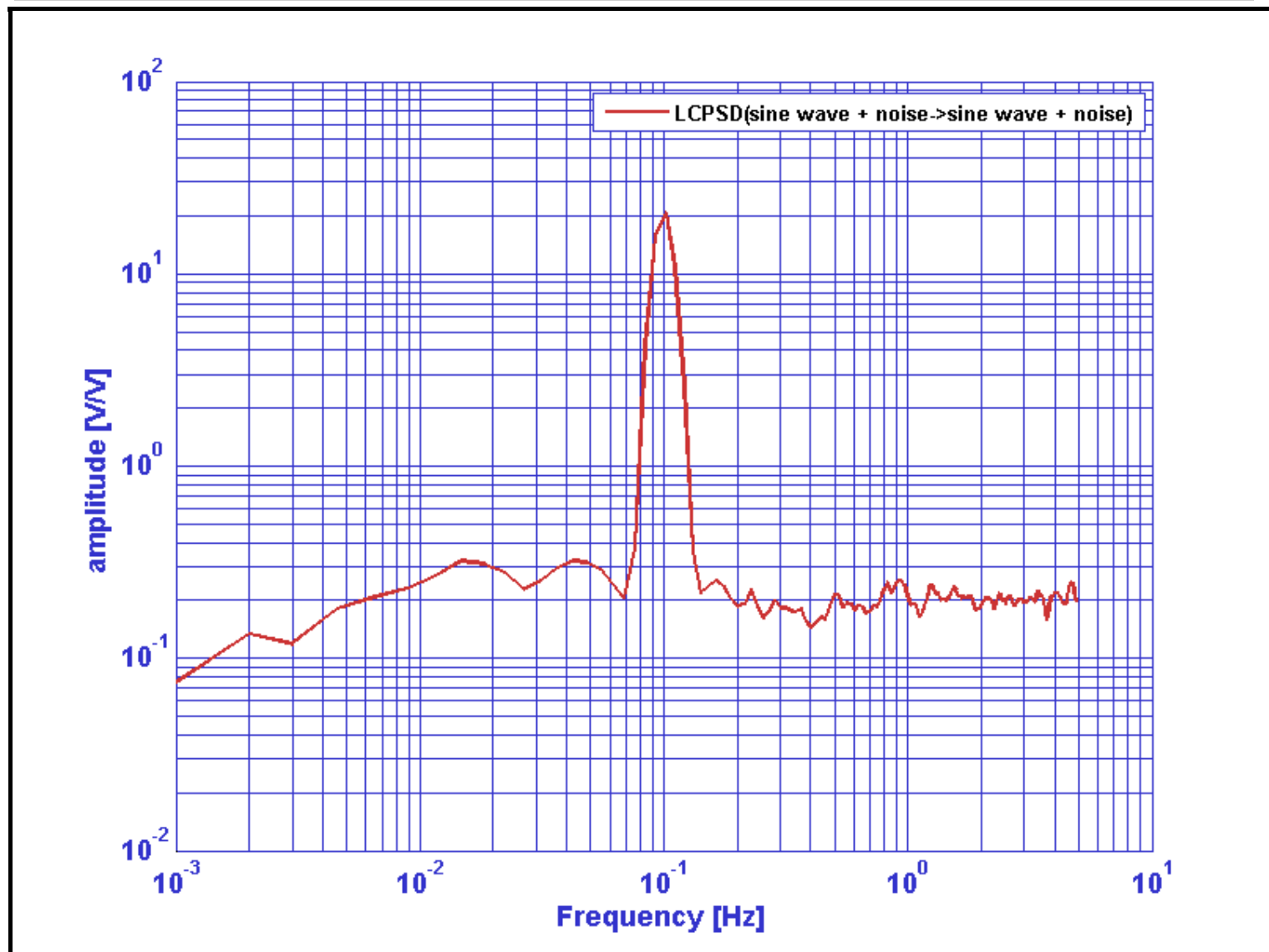
The length of the window is recalculated for each frequency bin, so the 'Win' parameter is used only to provide the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

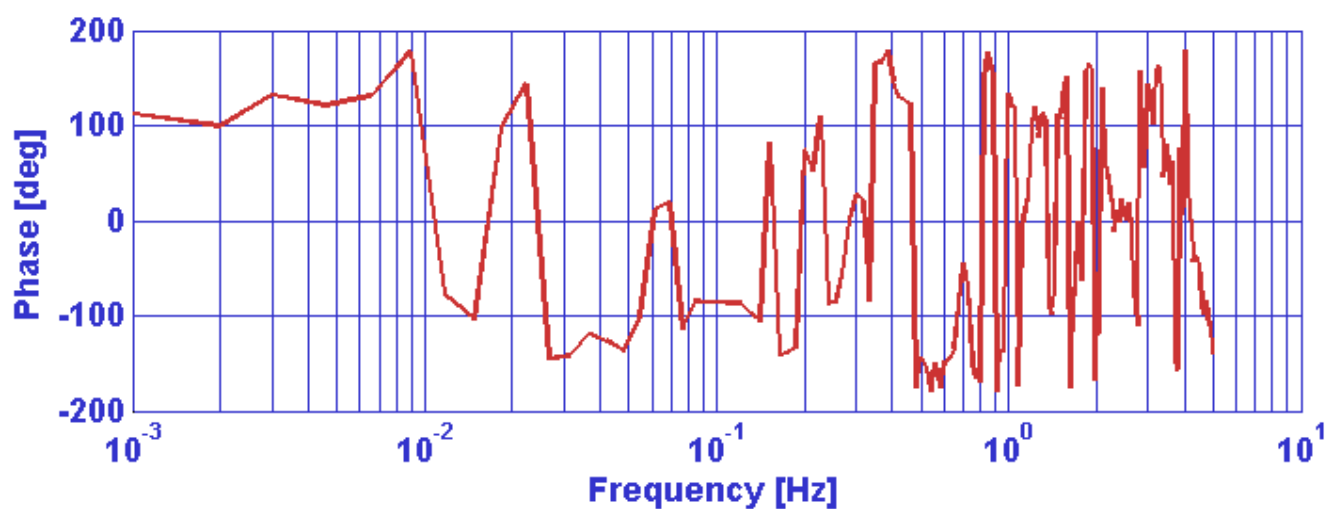
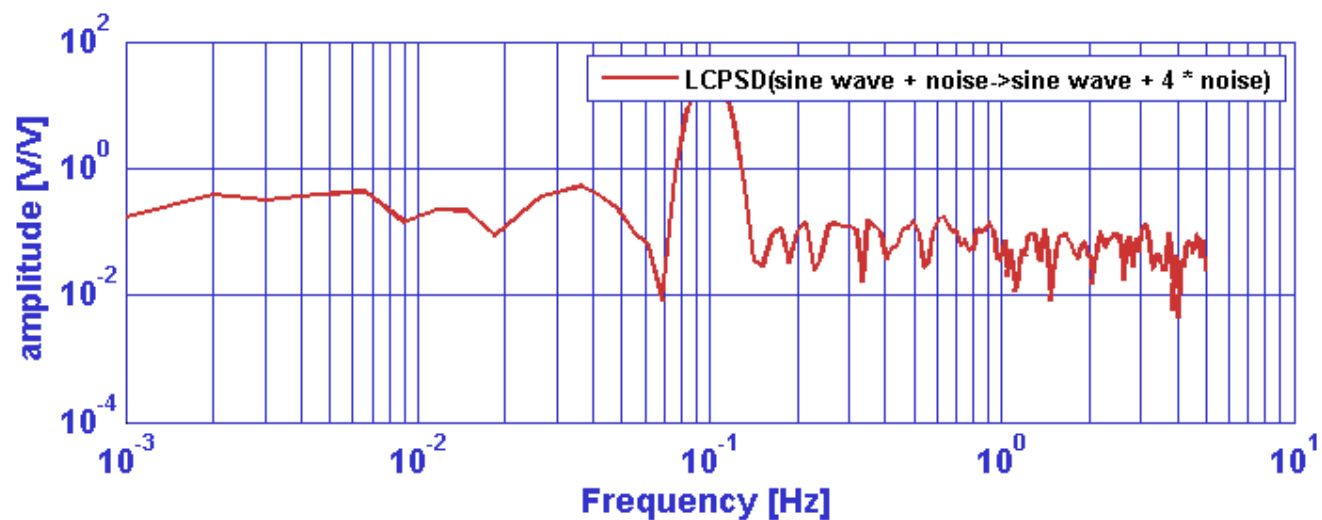
If the user doesn't specify the value of a given parameter, the default value is used. The function makes CPSD estimates between all input AOs. Therefore, if the input argument list contains N analysis objects, the output will contain $N \times N$ CPSD estimates. The diagonal elements will be S_{ai_ai} and will be equivalent to the output of `ltpda_lpsd(ai)`.

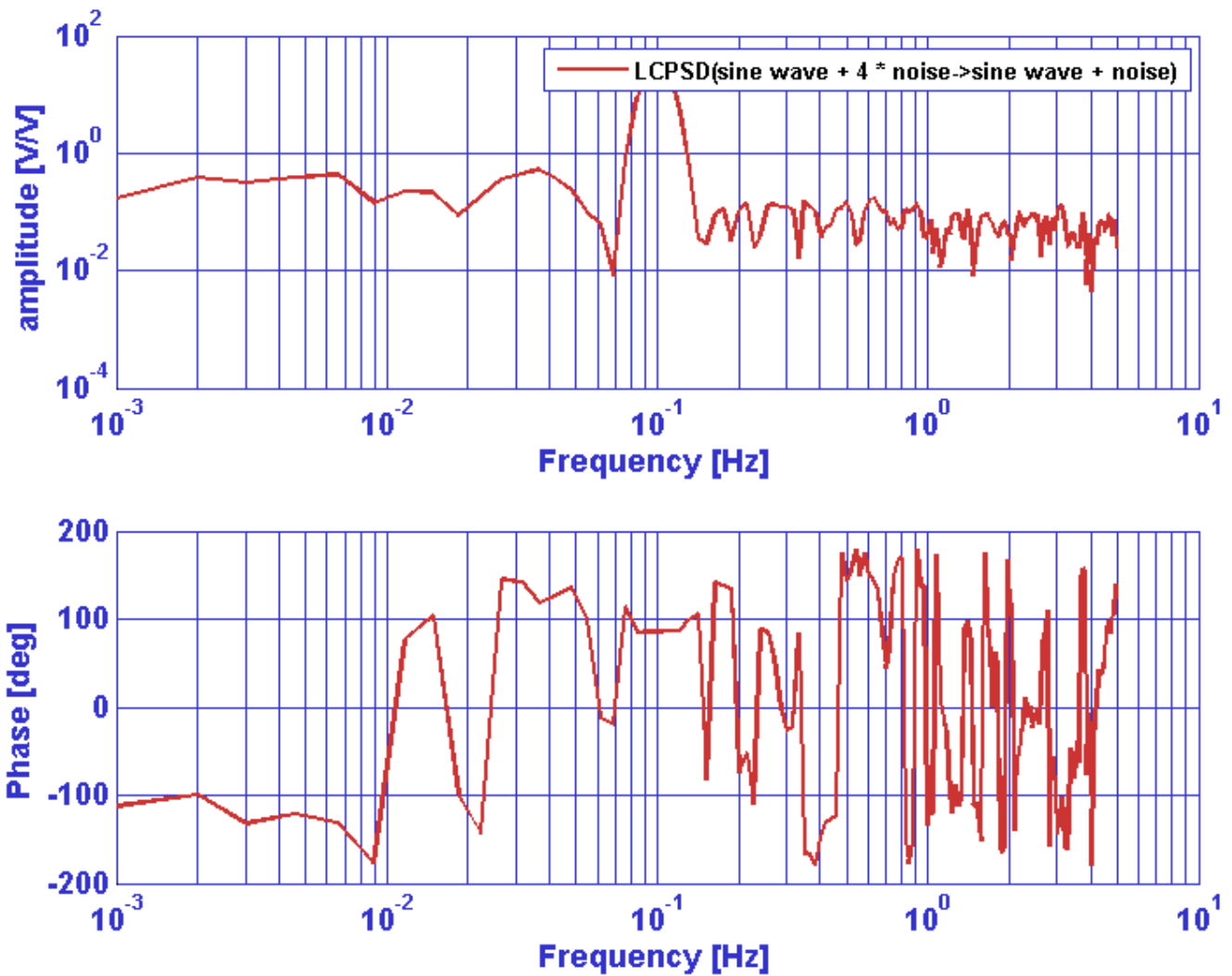
Example

Evaluation of the CPSD of two time-series represented by: a low frequency sinewave signal superimposed to white noise, and a low frequency sinewave signal at the same frequency, phase shifted and with different amplitude, superimposed to white noise.

```
x = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10)) + ...
ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
y = ao(plist('waveform','sine wave','f',0.1,'A',2,'nsecs',1000,'fs',10,'phi',90)) + ...
4*ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
pl = plist('order',2,'win',specwin(plist('name','BH92')));
z = ltpda_lcpsd(x,y);
iplot(z(1,1));
iplot(z(1,2));
iplot(z(2,1));
```







◀ Log-scale power spectral density estimates

Log-scale cross coherence density estimates ▶

©LTP Team

Log-scale cross coherence density estimates

Multivariate power spectral density on a logarithmic scale can be performed by the LPSD algorithm, which is an application of Welch's averaged, modified periodogram method, spectral density estimates are not evaluated at frequencies which are linear multiples of the minimum frequency resolution $1/T$ where T is the window length, but on a logarithmic scale. The algorithm takes care of calculating the frequencies at which to evaluate the spectral estimate, aiming at minimizing the uncertainty in the estimate itself, and to recalculate a suitable window length for each frequency bin.

[ltpda_lcohere](#) estimates the coherence of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectra, by multiplying it with a [spectral window object](#), and can be detrended by polynomial of time in order to reduce the impact of the border discontinuities. Detrending is performed on each individual window.

Syntaxis

```
b = ltpda_lcohere(a1,a2,a3,...,pl)
```

$a1$, $a2$, $a3$, ... are AOs containing the input time series to be evaluated. They need to be in a number $N \geq 2$. b includes the $N \times N$ output objects. The parameter list pl includes the following parameters:

- 'kdes' – the desired number of averages [default: 100]
- 'jdes' – the number of spectral frequencies to compute [default: $fs/2$]
- 'lmin' – the minimum segment length [default: 0]
- 'win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'order' – order of segment detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

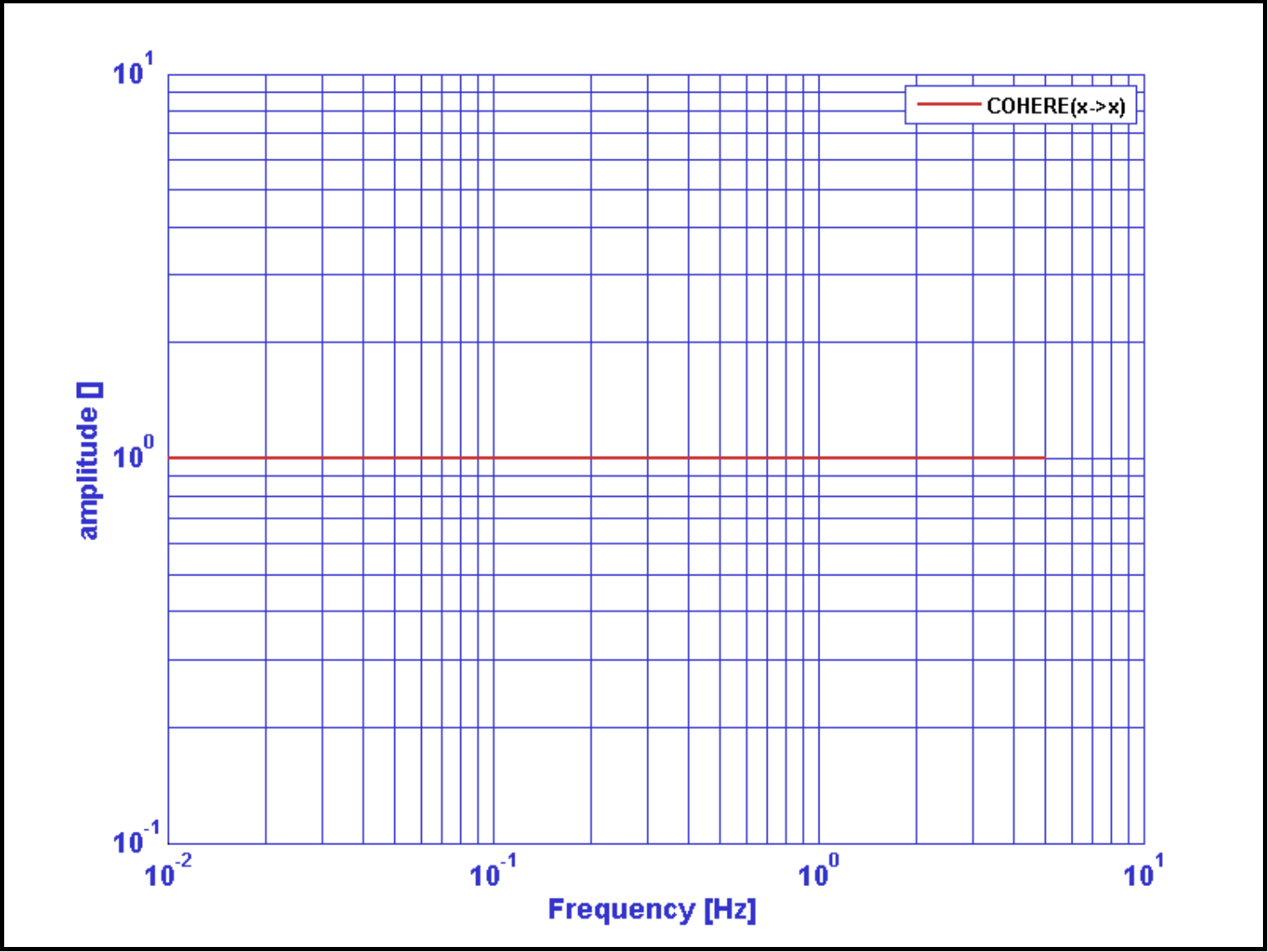
The length of the window is recalculated for each frequency bin, so the 'Win' parameter is used only to provide the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

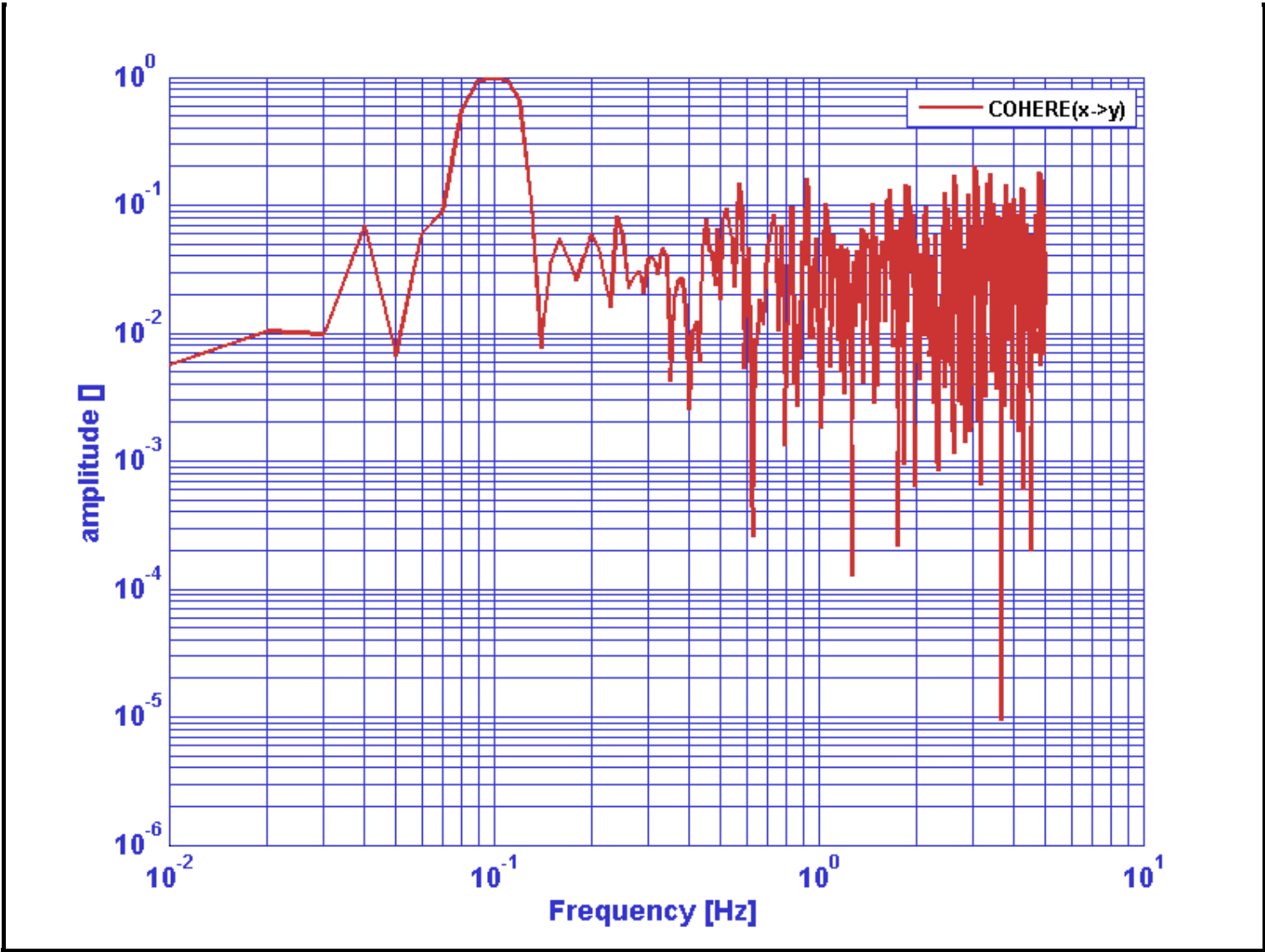
If the user doesn't specify the value of a given parameter, the default value is used. The function makes coherence estimates between all input AOs. Therefore, if the input argument list contains N analysis objects, the output a will contain $N \times N$ coherence estimates. The diagonal elements will be 1.

Example

Evaluation of the coherence of two time-series represented by: a low frequency sinewave signal superimposed to white noise, and a low frequency sinewave signal at the same frequency, phase shifted and with different amplitude, superimposed to white noise.

```
x = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10)) + ...
ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10)) + ...
ao(plist('tsfcn','t','nsecs',1000,'fs',10));
y = ao(plist('waveform','sine wave','f',0.1,'A',2,'nsecs',1000,'fs',10,'phi',90)) + ...
4*ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
pl = plist('win',specwin(plist('name','BH92')));
z = ltpda_lcohere(x,y,pl);
iplot(z(1,1));
iplot(z(1,2));
```





◀ Log-scale cross-spectral density estimates

Log-scale transfer function estimates ▶

Log-scale transfer function estimates

Multivariate power spectral density on a logarithmic scale can be performed by the LPSD algorithm, which is an application of Welch's averaged, modified periodogram method, cross-spectral density estimates are not evaluated at frequencies which are linear multiples of the minimum frequency resolution $1/\tau$ where τ is the window length, but on a logarithmic scale. The algorithm takes care of calculating the frequencies at which to evaluate the spectral estimate, aiming at minimizing the uncertainty in the estimate itself, and to recalculate a suitable window length for each frequency bin.

[ltpda_ltf](#) estimates the transfer function of time-series signals, included in the input AOs. Data are windowed prior to the estimation of the spectra, by multiplying it with a [spectral window object](#), and can be detrended by polynomial of time in order to reduce the impact of the border discontinuities. Detrending is performed on each individual window.

Syntax

```
b = ltpda_ltf(a1,a2,a3,...,pl)
```

$a_1, a_2, a_3, \dots$ are AOs containing the input time series to be evaluated. They need to be in a number $N \geq 2$. b includes the $N \times N$ output objects. The parameter list pl includes the following parameters:

- 'kdes' – the desired number of averages [default: 100]
- 'jdes' – the number of spectral frequencies to compute [default: $f_s/4$]
- 'Lmin' – the minimum segment length [default: 0]
- 'win' – a specwin window object [default: Kaiser –200dB psll]
- 'olap' – segment percent overlap [default: taken from window function]
- 'order' – order of segment detrending
 - -1 – no detrending
 - 0 – subtract mean [default]
 - 1 – subtract linear fit
 - N – subtract fit of polynomial, order N

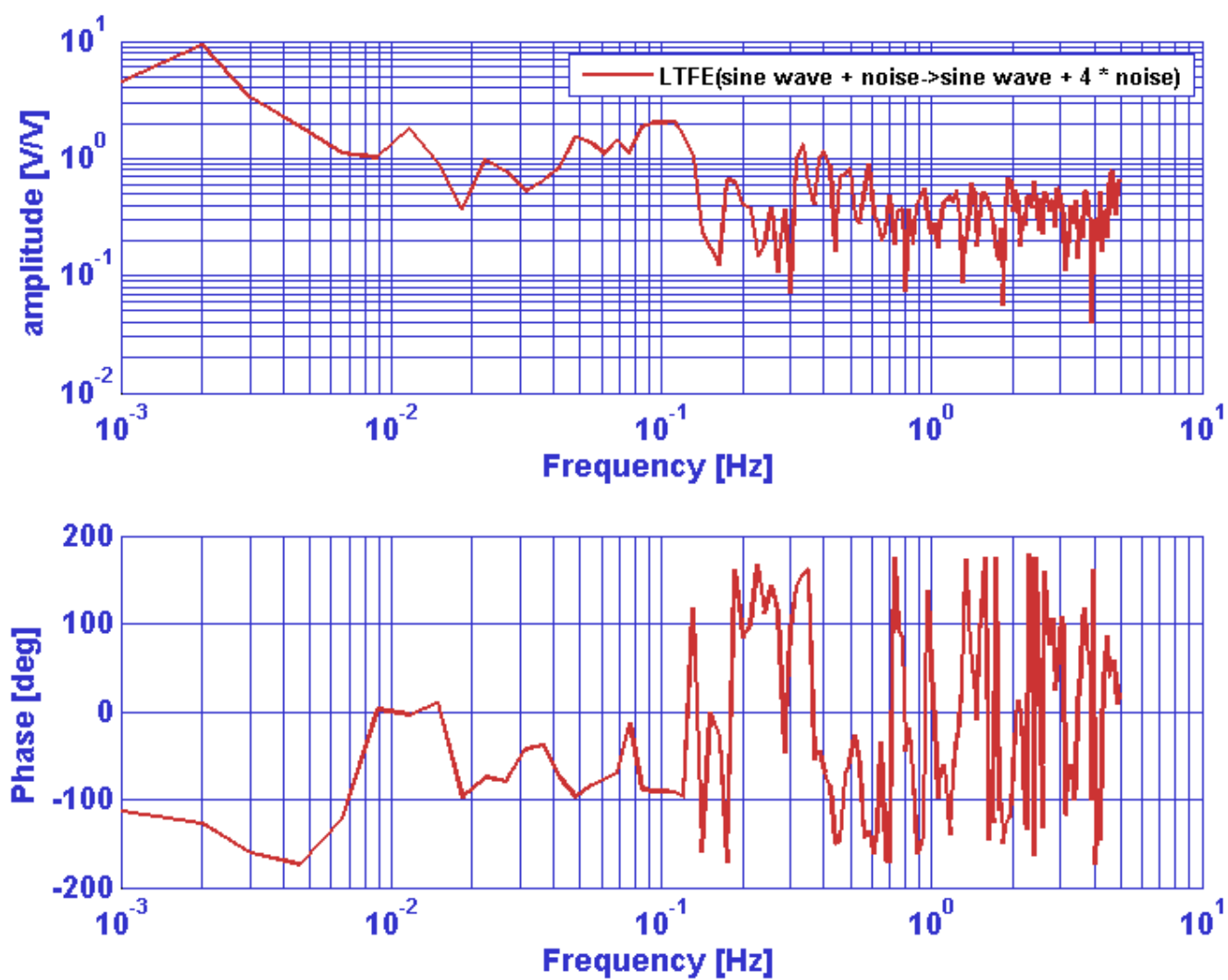
The length of the window is set by the value of the parameter 'nfft', so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

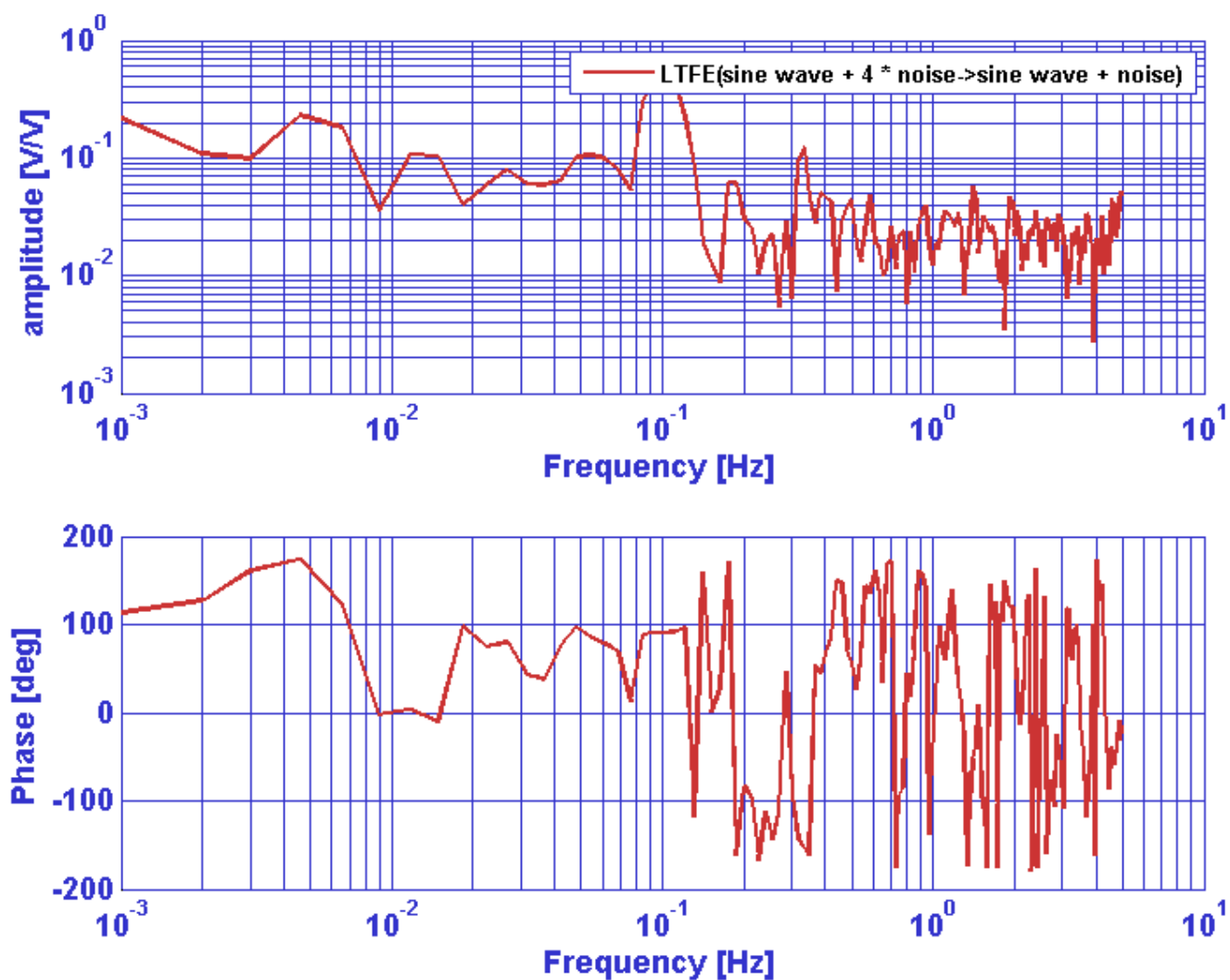
If the user doesn't specify the value of a given parameter, the default value is used. The function makes transfer functions estimates between all input AOs. Therefore, if the input list contains N analysis objects, the output a will contain $N \times N$ TFE estimates. The diagonal elements will be 1.

Example

Evaluation of the transfer function between two time-series represented by: a low frequency sinewave signal superimposed to white noise, and a low frequency sinewave signal at the same frequency, phase shifted and with different amplitude, superimposed to white noise.

```
x = ao(plist('waveform','sine wave','f',0.1,'A',1,'nsecs',1000,'fs',10)) + ...
    ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
y = ao(plist('waveform','sine wave','f',0.1,'A',2,'nsecs',1000,'fs',10,'phi',90)) + ...
    4*ao(plist('waveform','noise','type','normal','nsecs',1000,'fs',10));
z = ltpda_ltf(x,y,plist('win',specwin(plist('name','BH92'))));
ipplot(z(1,2));
ipplot(z(2,1));
```





◀ Log-scale cross coherence density estimates

Fitting Algorithms ▶

©LTP Team

Fitting Algorithms

The following sections describe special tools for data fitting implemented in the LTPDA toolbox.

- [Polynomial Fitting](#)
- [Time Domain Fit](#)

©LTP Team

Polynomial Fitting

[polyfit.m](#) overloads the polyfit() function of MATLAB for Analysis Objects.
The script calls the following MATLAB functions:

- polyfit.m
- polyval.m

Usage

```
% CALL:          b = polyfit(a, pl)
%
% Parameters:    'N'      - degree of polynomial to fit
%                'coeffs' - (optional) coefficients
%                formed e.g. by [p,s] = polyfit(x,y,N);
```

The MATLAB function polyfit.m finds the coefficients of the polynomial $p(x)$ of degree N that fits the vector 'x' to the vector 'y', in a least squares sense.
After this in the script [polyfit.m](#) the function polyval.m is called, which evaluates the polynomial of order 'N' according to these coefficients.
Using the output of polyval.m the fitted data series is created and outputted as analysis object.

Time domain Fit

[ltpda_timedomainfit.m](#) uses the MATLAB function `lsconv.m` to fit a set of time-series AOs to a target time-series AO. It gives back a set of fitting coefficients.

One can now subtract the fitted time series from the original one – the target – and produce a new time series by calling [ltpda_lincom.m](#) with the calculated coefficients. This function does a linear combination of the inputted coefficients and analysis objects and subtracts the result from the target analysis object, which has to be the first input parameter.

An example:

```
coeffsA0 = ltpda_timedomainfit(target, ts1, ts2, ts3, ts4);  
%% Make linear combination  
x12ns = ltpda_lincom(target, ts1, ts2, ts3, ts4, coeffsA0);
```

`x12ns` represents the noise subtracted target analysis object. With noise here the fitted time series is meant. That is the linear combination of the coefficients `coeffsA0` and the time series objects `ts1` to `ts4`.

The number of time or frequency series analysis objects is variable. The first is always taken as target object.

Graphical User Interfaces in LTPDA

LTPDA has a variety of Graphical User Interfaces:

- [The LTPDA Launch Bay](#)
- [The LTPDA Analysis GUI](#)
- [The LTPDA Repository GUI](#)
- [The pole/zero model helper](#)
- [The Spectral Window GUI](#)
- [The constructor helper](#)
- [The LTPDA object explorer](#)
- [The quicklook GUI](#)

The LTPDA Launch Bay

The LTPDA Launch Bay GUI allows quick access to all other GUIs in LTPDA. To start the Launch Bay:

```
>> ltpdalauncher
```

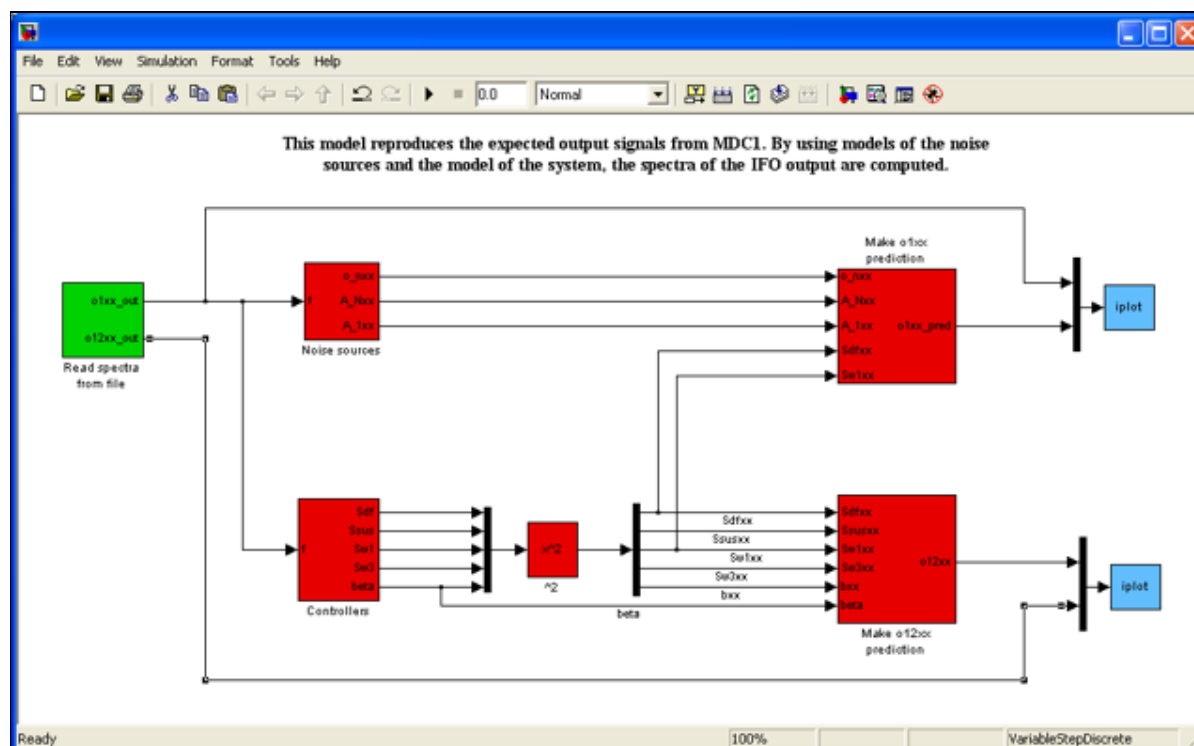


The Launch Bay is automatically started from the `ltpda_startup` script.

The LTPDA Analysis GUI

Introduction

The LTPDA package contains a sophisticated graphical programming interface which allows the user to construct and execute data analysis pipelines entirely graphically.

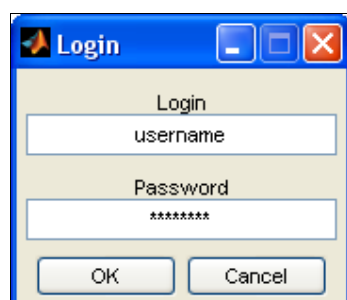


The Simulink drawing interface is used to construct the analysis pipelines. The blocks that compose the analysis pipelines are drawn from the LTPDA blockset. The LTPDA Analysis GUI then executes the commands represented by each block in the appropriate order in order to achieve the desired analysis pipeline.

To start the LTPDA GUI use the command `ltpdagui`.

```
>> ltpdagui
```

If there's no open LTPDA GUI window, the user will be asked to log in:



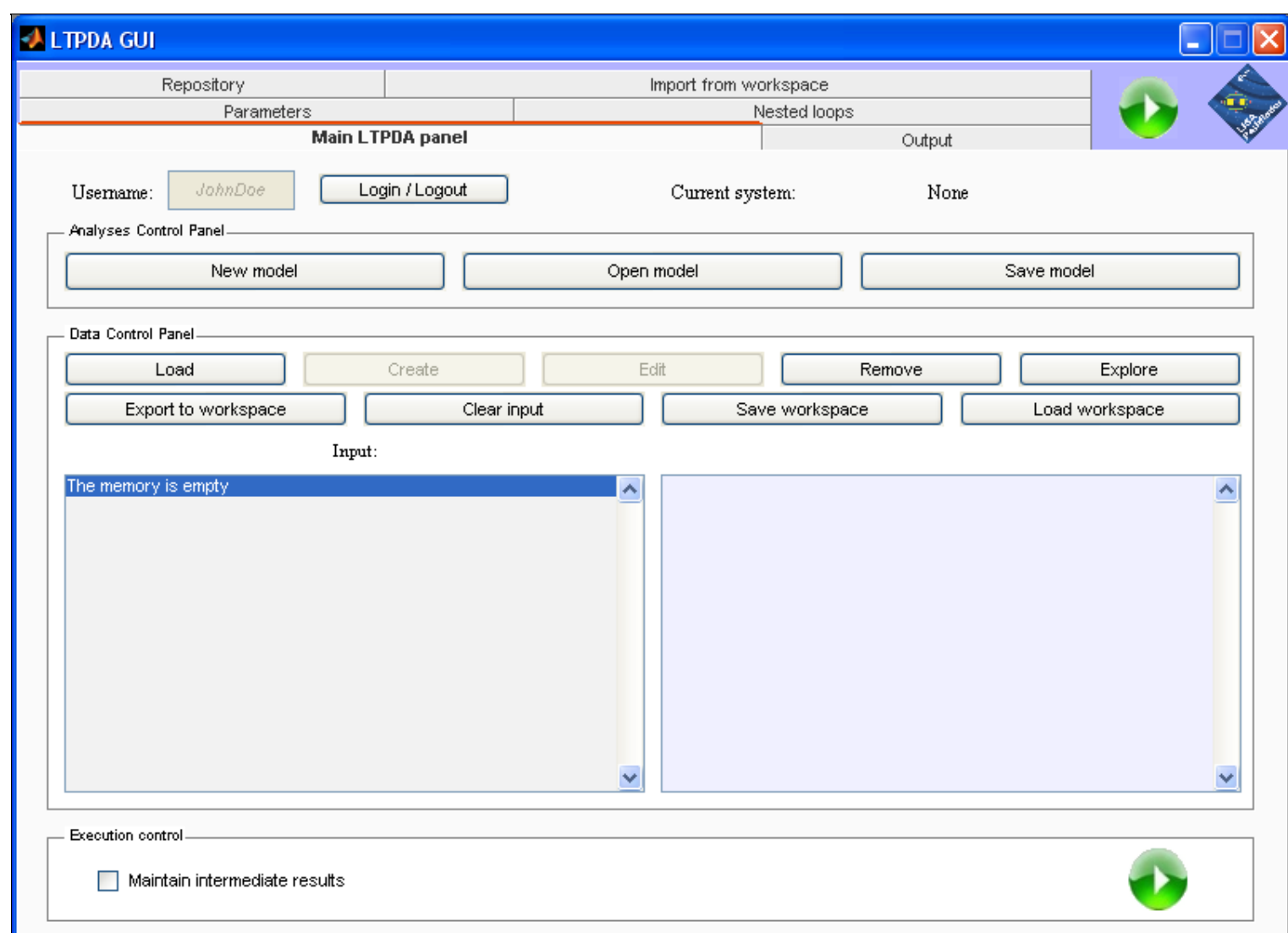
These data are not mandatory, as long as the user do not want to connect to a LTPDA Repository, for example to retrieve previous analyses or to send in his own work. If the user just wants to use the software to compile trials or personal analyses, not meant to be shared with others, he can leave these fields empty: the GUI will identify him as anonymous ('JohnDoe'), and this name will be stored into every produced model.

On the contrary, if the GUI has to produce shared results, i.e. if the user want to submit/retrieve from an LTPDA Repository, these login and password are those necessary to access the repository database. The user must possess a valid account, which can be requested to the database administrator.

The username will be used also to ensure better traceability, storing the name itself into a short description created using the LTPDA GUI.

However, it's possible at anytime to log off and re-log in, using different data, so it will be possible to type in valid credentials to access a repository only when needed.

The first appearance of the GUI is:



The upper part of the window lists all the possible panels: clicking on the corresponding tab the proper panel will be drawn below.

The help associated to each panel will be dealt into different pages.

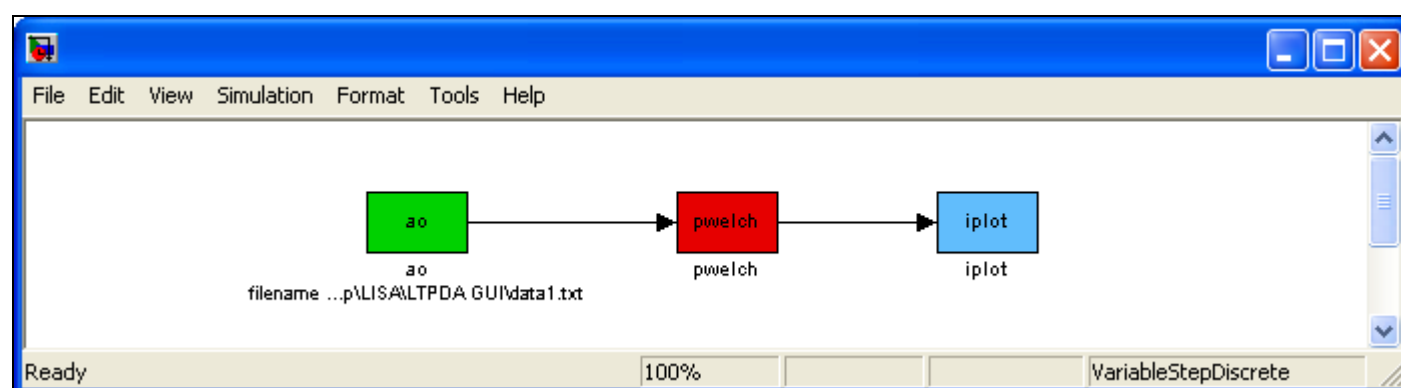
Basics

Data handling in Simulink

Since SIMULINK can only send among blocks a limited set of object types (that is, inside Simulink lines can only travel numbers, vectors or Simulink defined signals), **the main concept about LTPDA GUI and LTPDA Simulink models is that what's passed among blocks is not a true LTPDA object (an AO, a filter, a specwin or so), but rather a pure number, intended as an index inside a given array of objects.**

Every analysis block in a LTPDA model produces the output of its calculation, then stores this output into a global array – which will be accessible by all following functions/blocks – and send back to Simulink just the index of this newly added object inside the array.

For example:



The first block load create an AO from a file on disk. This AO is saved in a global array: assuming the array was empty, this is the 1st object, so '1' is the number sent by the first block into Simulink.

The second block receives this '1': the corresponding (1st) object is retrieved from the array, then the analysis function (`pwelch`) is applied. The output is appended back into the array, as 2nd element. So, '2' is the content of the signal leaving the second block.

The third block receives '2', retrieves the corresponding (2nd) object from the array and produces a plot of it.

Thus, during the analysis cycle the array of objects becomes populated: unless differently set, all the temporary objects created during the analysis will be canceled at the end of the analysis itself.

The LTPDA GUI is based on two global arrays:

- The aforementioned **'Input array'**, which encloses all the LTPDA objects created by the user (for example, loading from file) and the objects generated during the analysis.

This is shown into the main panel of the LTPDA GUI.

- The '**Output array**', meant as a way to separate primary objectives of the analysis from the other – working – objects, contained in the Input array.
This is shown into its own panel, the Output panel.

The only way to save an object into the Output array is using the proper 'Send to Output' block, which will copy the object passed from the Input to the Output array.

Parameters

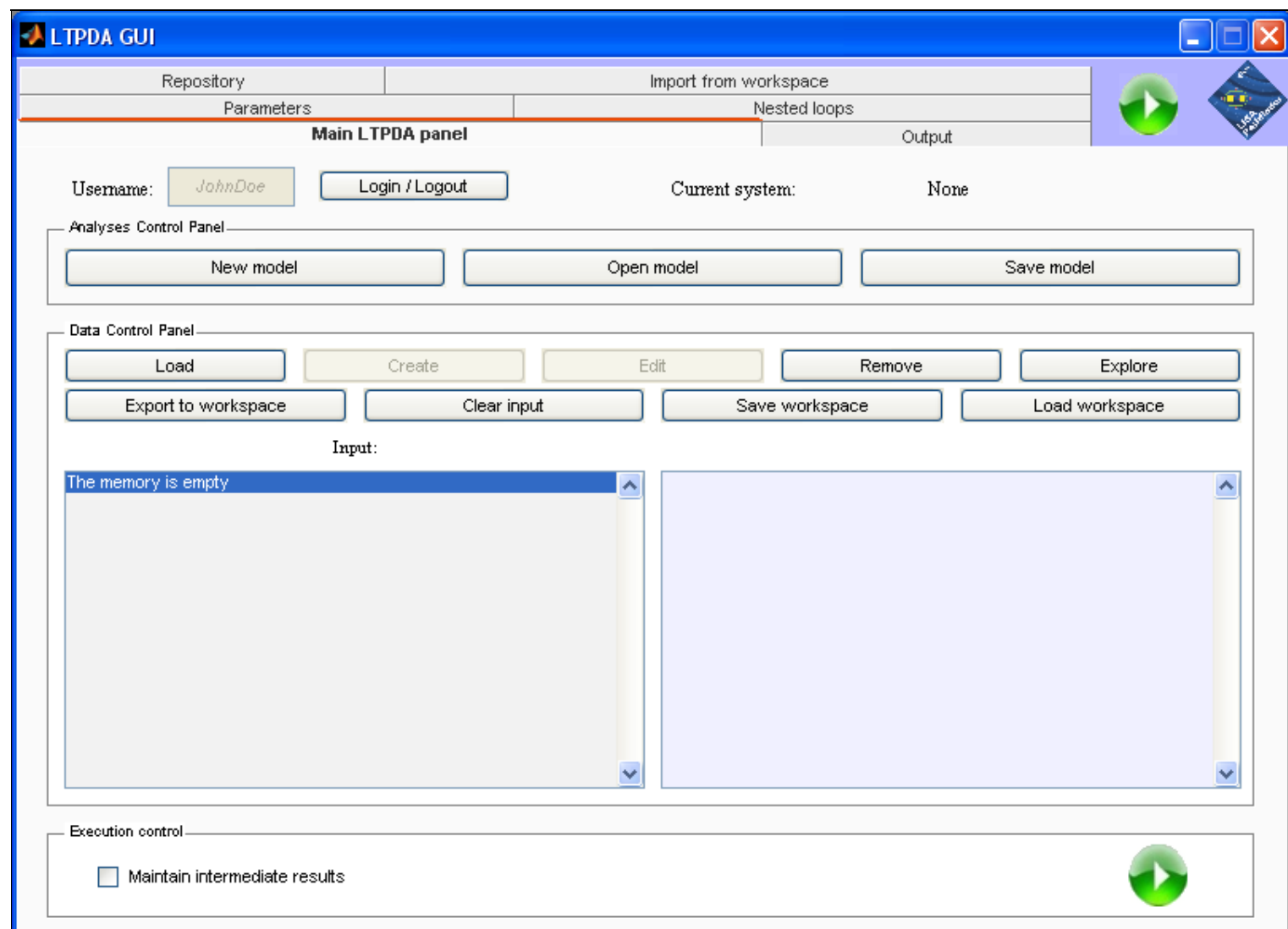
In order to enclose all the parameters of each block together with the Simulink model file, each parameters list is converted into a string and saved into the Block Description, a specific field of every block in Simulink.
Every time the user selects a block inside a valid LTPDA model the LTPDA GUI will automatically read this description to convert it back to a proper plist. 'Valid' LTPDA models are those created by the GUI, thus containing the short annotation with the author's name, time and date of creation and so on.

For more information on parameters handling, see the paragraph regarding the Parameters panel.

◀ The LTPDA Analysis GUI	Main panel ▶

Main panel

The appearance of the main (first) panel is:



Access and selection control

In the upper part of the panel is shown the username typed in by the user when the LTPDA GUI was started.

If the login field was left empty the GUI will consider the anonymous 'JohnDoe' user; the password field in this case has been ignored, was it empty or not.

This username is used to identify the current user of the system, and it's stored into the annotation of every analysis created by the GUI, clicking on the 'New model' button.

Is possible to change the active user clicking on the 'Login/Logout' button; this is to be used when:

- The username do not respect the current active user of the system.
- The username and/or password typed in aren't valid credentials to connect a LTPDA Repository, and the user wants to submit/retrieve something.

The current user is leaving the computer, to prevent other users to submit by mistake personal objects to a repository with a wrong account; this would make difficult to keep results traceable.

On the right the GUI shows the currently active model, if the selected model is a valid LTPDA one. 'Valid' LTPDA models are those created by the GUI, thus containing the short annotation with the author's name, time and date of creation and so on. If the selected model isn't recognized by the GUI 'None' will be shown here, and no selection control will be performed upon what the user clicks inside that model.

Analysis control panel

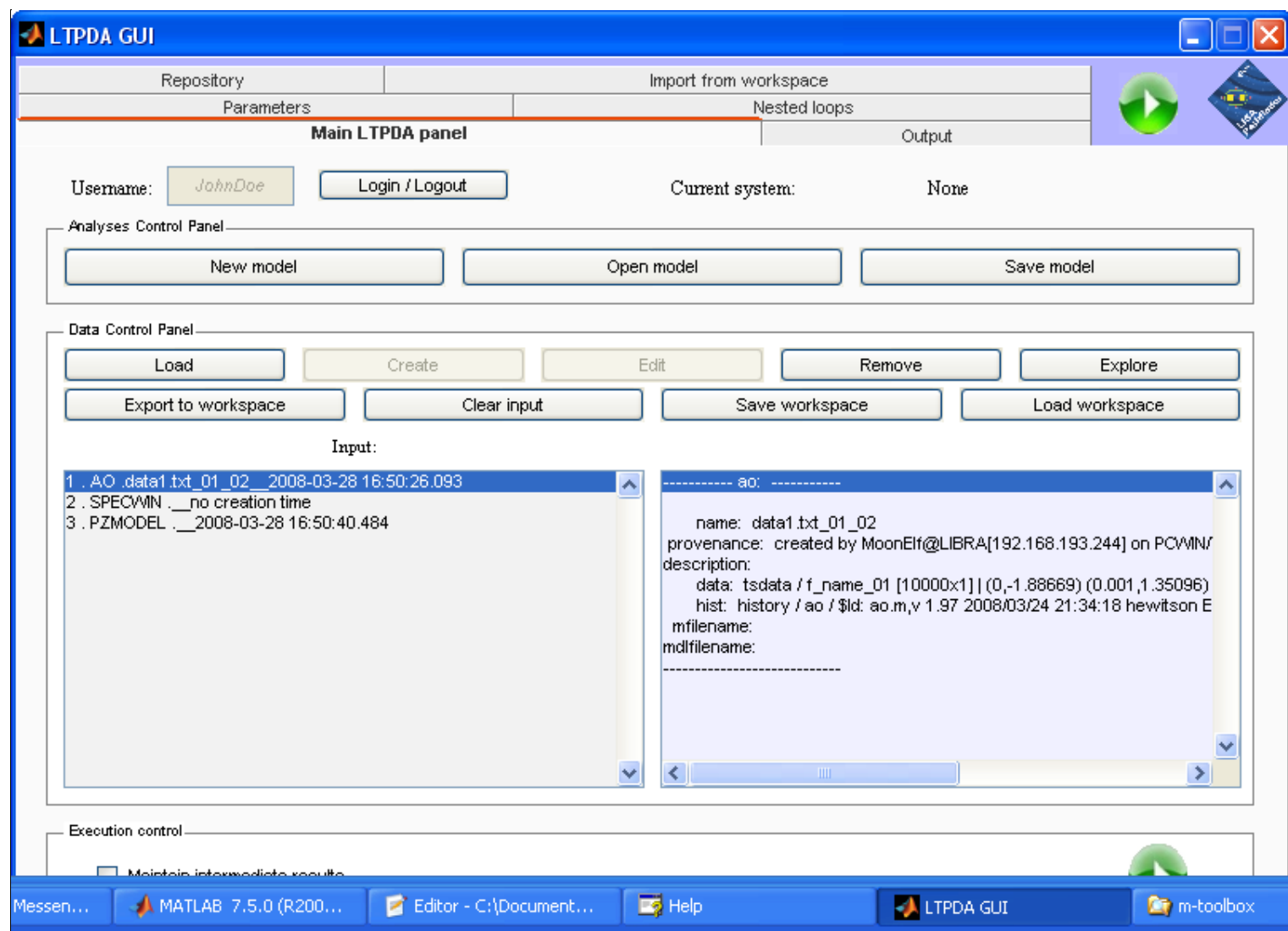
This contains 3 buttons, all devoted to the control of LTPDA Simulink model files.

- **'New model'**: it creates an empty valid LTPDA Simulink model. The annotation inside will be created automatically to respect the current user, time and date, IP address, operative system running and MATLAB/LTPDA versions.
- **'Open model'**: similarly to the common 'open' command in Simulink, this button loads an existing Simulink file from disk. If the file contains a valid LTPDA model, then the GUI will perform the selection check when it's selected.
- **'Save model'**: lets the user to save the currently selected model. Unlike the common 'save' command in Simulink this will prompt the user with the assigned filename (username + time/date of creation) and the predefined folders for analyses, if any was set in the `ltpda_startup` file. Anyway, the user can freely rename this to any name, withstanding the common Simulink filename limitations.

Data control panel

Containing 9 buttons and the summary of the Input array, this part is devoted to the control of Input objects (i.e., data created by the user or generated inside a previous analysis run).

- **'Load'**: to load data from ASCII file (to be converted into an AO), or from .MAT or .XML files. If the selected MAT or XML file contains any valid LTPDA object, this is imported into the Input array.
- **'Create'**: *Reserved for future use.*
- **'Edit'**: *Reserved for future use.*
- **'Remove'**: to remove the selected object(s) from the Input array. The numbering of the remaining objects is not altered, to prevent problems in the indexing of objects into the analysis models opened.
- **'Explore'**: to pass the selected object(s) to the `ltpda_explorer`, to explore them into an expandable and browsable window.
- **'Export to workspace'**: to save the selected object(s) into the MATLAB workspace. The name of these objects will be automatically set.
- **'clear input'**: to clear the entire Input array.
- **'Save workspace'**: even if this button is situated only inside the main panel, the `Save workspace` function it recalls affects not only the Input array, but the Output array too. It's meant as a solution to capture the entire current status of the LTPDA GUI, saving it into a .MAT file on disk.
- **'Load workspace'**: as the `Save workspace` button, it lets the user to load from a .MAT file on disk the entire saved status of the Input and Output arrays. The loaded contents are appended to the existing arrays in memory.



The central part of the window contains two boxes: on the left a list representing the contents of the Input array, on the right the details of the object currently selected in the list aside.

Note this will list only LTPDA objects: if any function/block in the analysis produced as output a simple numerical vector or matrix it won't be visible here.

The single click is used to select, to have a look of the details of a single object in the Info panel on the right.

Double clicking on the list will add to the active LTPDA Simulink model (if any is selected) a "Object from list" block, so to import the object double-clicked in the given model.

Execution control panel

At the bottom of the window there are:

- **'Maintain intermediate results':** enabling this ALL the outputs produced during the analysis (i.e., the output of every block) will be kept at the end of the analysis itself. This is meant as a way to verify intermediate results without the need to enable the 'Maintain' option for every single block, but of course should be used with care, since it will make the Input array very long and populated in a short time.
- **'start button':** this button starts the execution of the analysis. Till the end of the analysis it will turn to red. Although the common Simulink 'start simulation' button will correctly execute the model, the user should anytime use just the button enclosed in the GUI, which fulfills many extra purposes; for example, it will update the Input array list in the main panel, removing all

the intermediate results (whenever not differently set), it will open the progress bar window to represent the analysis proceeding, and it will add terminators in the model wherever necessary.

To make easier the start of the analysis, a direct copy of this button is shown in the upper right corner of the window, outside the first panel. This is visible from every panel, so it will be possible to start the execution without the need to open the main panel first.

◀ Basics

Parameters ▶

©LTP Team

Parameters

The parameters panel will be empty if:

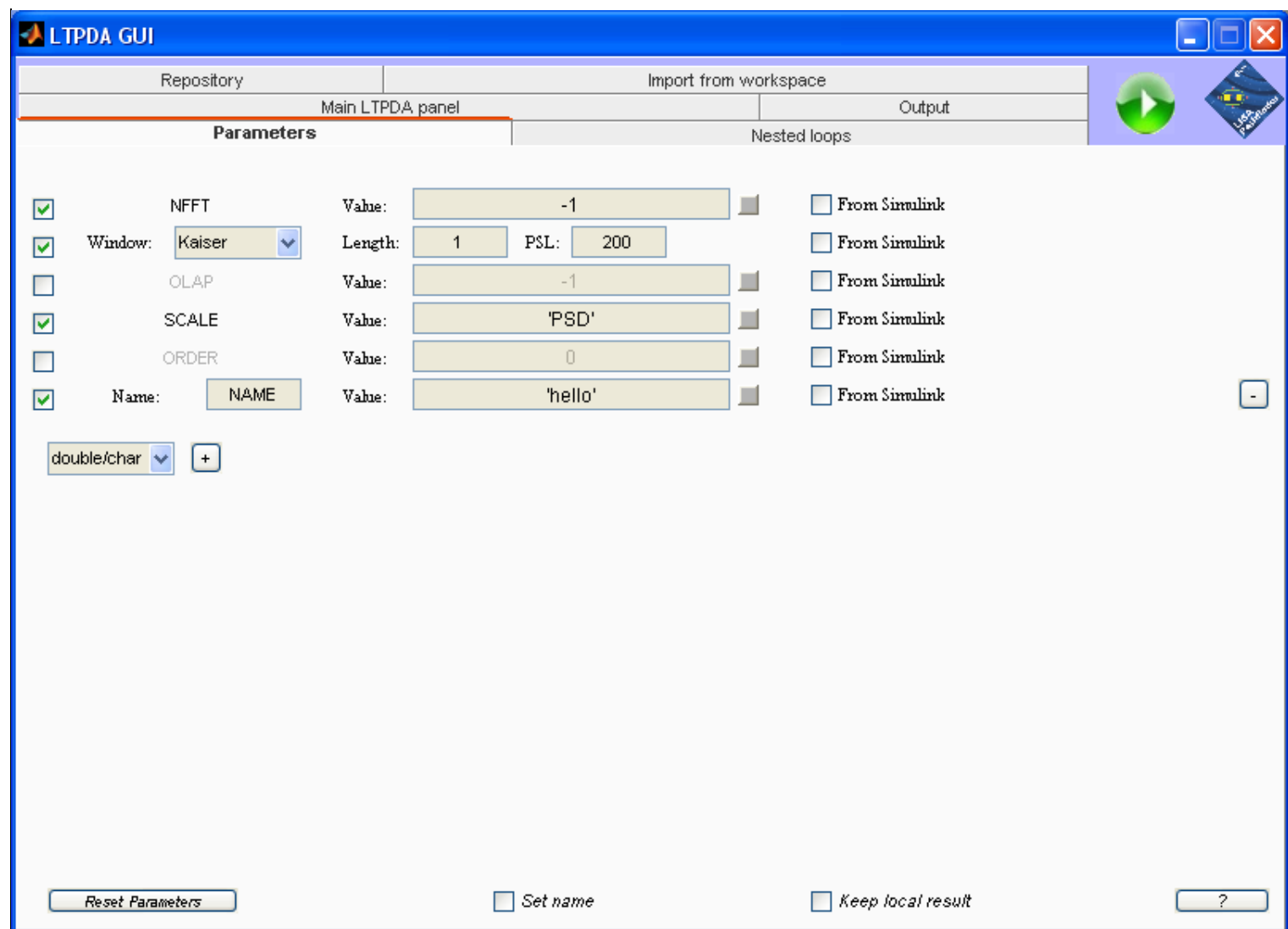
- No block is currently selected.
- The currently selected Simulink model is not a valid LTPDA model.
- The selected block it's not a valid LTPDA block (i.e., it do not contain a proper LTPDA function which answers the call for parameters).

Otherwise, the contents of the parameters panel will change accordingly to the user selection in the active LTPDA Simulink model.

- If a **function/method** block is selected, the panel will show the proper corresponding parameters.
- If a **Object from list** block is selected, the panel will show the referenced index (that is the only parameter accepted).
- If a **Mux/Demux** block is selected, the panel will show the number of inputs/outputs.
- If a **From** block is selected, the panel will present the 'Find origin' button.

Function/method parameters

If a valid LTPDA function or method block is selected (inside a valid LTPDA Simulink model), the GUI will first ask the inner function or method for its required parameters; the panel then will be built accordingly.



The window is built automatically: **every line corresponds to a different parameter**, among those required by the selected function.

Inside every line, then, the structure is obviously similar, so different parameters can have the same fields and boxes.

Required or added parameters

In the construction of the panel the GUI will consider whether the parameter was among those required by the function, or if the parameter was added by the user. In this latter case it will be passed to the function, which can use it (if somehow expected, for example if it is a possible optional parameter) or ignore it, if just unexpected. The GUI won't change the behavior of the inner function, so it's up to the user to be aware of what can be passed to be used by every function.

In the parameters panel the difference among required and added parameters is shown on the far right: added parameters have a small '-' button, which lets the user to remove it; required parameters have no such button, since the corresponding parameter was automatically required by the function and not added by the user.

Please note that a parameter can also be disabled (see 'Enable checkbox', below): since a disabled parameter won't be passed to the function, there's no difference in removing an added parameter or just disable it. The only difference of course is a disabled parameter can be re-enabled in a second time, while a removed parameter has to be added and set over again.

Enable checkbox

On the left, at the beginning of every line/parameter, there is a checkbox which lets the user to enable or disable the corresponding parameter. If it's enabled the parameter will be passed to the function, so it's up to the user to set the proper value. If it's disabled, on the contrary, it won't be part of the plist passed to the function, which thus will use the default value (if any is expected).

The central part of the window is built line by line to reflect each parameter, so it can differ a lot.

The screenshot shows the LTPDA GUI window with a blue title bar. The main area is divided into two sections: 'Parameters' on the left and 'Nested loops' on the right. The 'Parameters' section contains a list of parameters with checkboxes and input fields. The 'Nested loops' section contains a list of parameters with checkboxes and input fields. The 'Parameters' section includes parameters like NFFT, Window, OLAP, SCALE, ORDER, Name, POLES, ZEROS, Gain, Poles, Zeros, fs, Order, fc, Ripple, and a 'double/char' parameter. The 'Nested loops' section includes parameters like From Simulink. The 'Parameters' section also has a 'Load' button for each parameter. The 'Nested loops' section has a 'From Simulink' checkbox for each parameter. The 'Parameters' section has a 'Reset Parameters' button at the bottom left. The 'Nested loops' section has a 'Set name' checkbox and a 'Keep local result' checkbox at the bottom right. The 'Parameters' section has a 'double/char' parameter with a dropdown menu and a '+' button. The 'Nested loops' section has a '?' button at the bottom right.

Parameter name

Wherever expected, the first element on the left, aside the Enable checkbox, is the parameter name. For **double/char** parameters this means the parameter key; note that if the parameter was required by the function, the name is fixed (see parameter 1,3,5,6 in the previous image), while for added parameters it can be changed (see parameter 7).

In case of an added parameter the GUI will show both a text 'Name:' and an edit field, where the user can type in the parameter key itself.

For **specwin** parameters the key is fixed, so the GUI shows directly the name of the window type ('Kaiser', 'Rectangular',...).

Similarly, for **Poles and Zeros** parameters the key is fixed, so even for added parameters 'Poles' and 'Zeros' are shown and cannot be altered.

Since the key is fixed also for **MIIR and PZmodel** parameters, the GUI shows directly the inner parameters to construct these objects; note that this is just a way to build a MIIR object, typing in

directly the plist. To build it in a different way, for example retrieving from the Repository, must be used a proper Constructor block, setting the parameters to be retrieved from Simulink (see 'From Simulink checkbox', below) and connection the Constructor block to the function port having as a name the corresponding parameter's name.

Parameter value

The following field lets the user to type in the parameters value. This is true for every 'value:' field in the parameters panel.

Please note that often is possible to have a brief help text in the tooltip string, just moving the mouse pointer over the edit field and waiting a second.

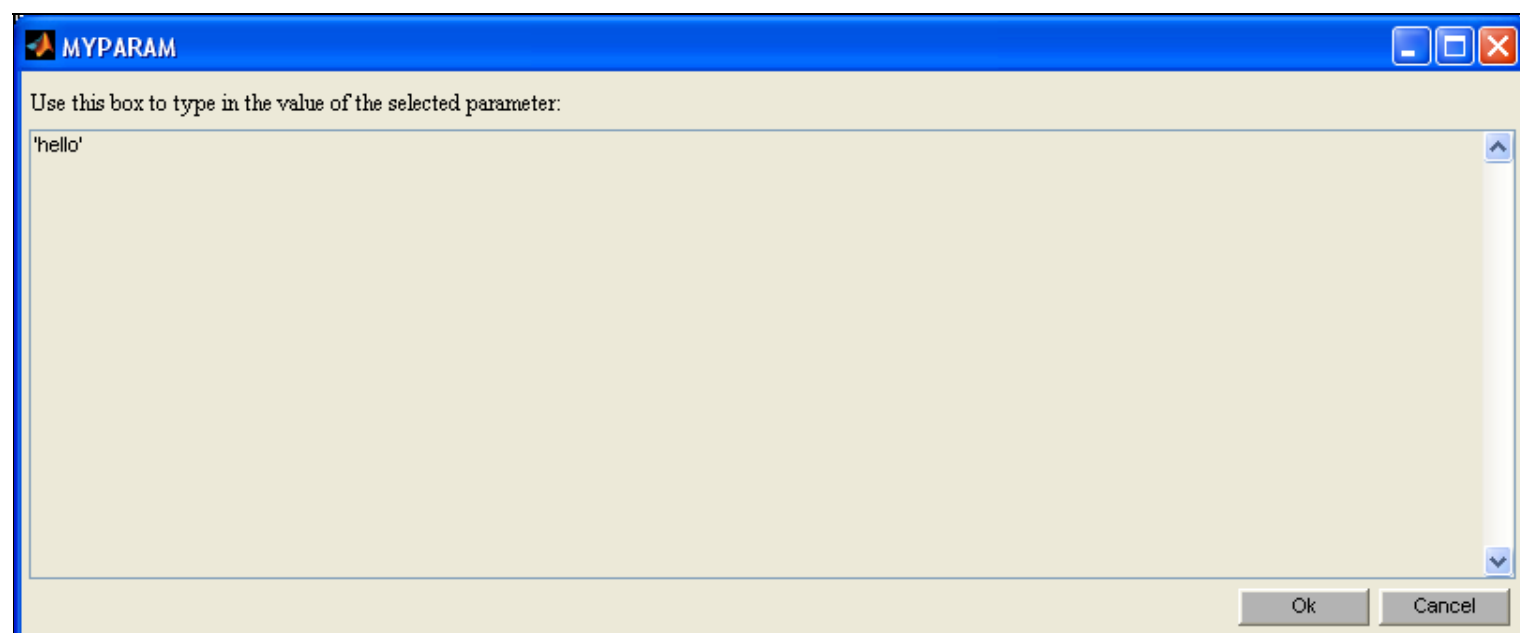
This will explain for example the meaning for the different edit fields in a specwin parameter, or it will explain how to type in poles and zeros into their corresponding edit fields.

Inside an edit field the user can type in using the traditional MATLAB syntax:

- **10**, will be considered a pure number. The parameter's value will be of course the number itself.
- **[1 2 3]**, among square brackets, will be converted into a vector (or matrix). Similarly, using **{}** will create a cell array. The parameter's value will be a vector, a matrix or a cell array.
- **'Hello'**, among quotes, will be considered a string, and it won't be evaluated. The parameter's value will be a string.
- **rand(1)**, without quotes, will be recognized as a command and it will be evaluated immediately. The parameter's value will be the output of the evaluated string.

Expand edit field button

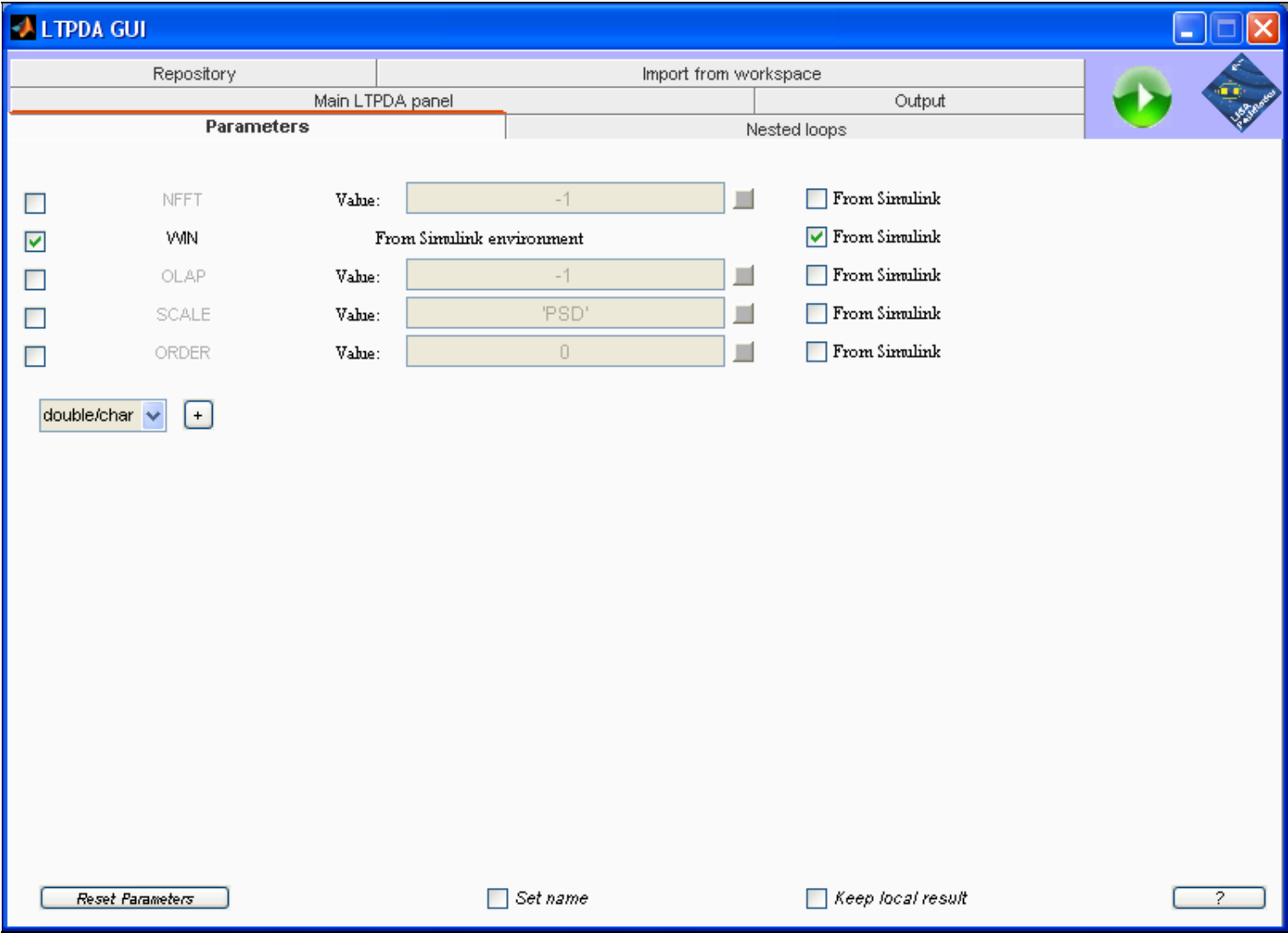
The small gray button aside each parameter's value edit field, for double/char parameters, will open up a window with a bigger edit field, where the user can type in the value more comfortably than in the small one in the panel. This is particularly handy for those case when the parameter's value is a long and complicated command or vector, which can't fit easily in the small space given in the panel.



From Simulink checkbox

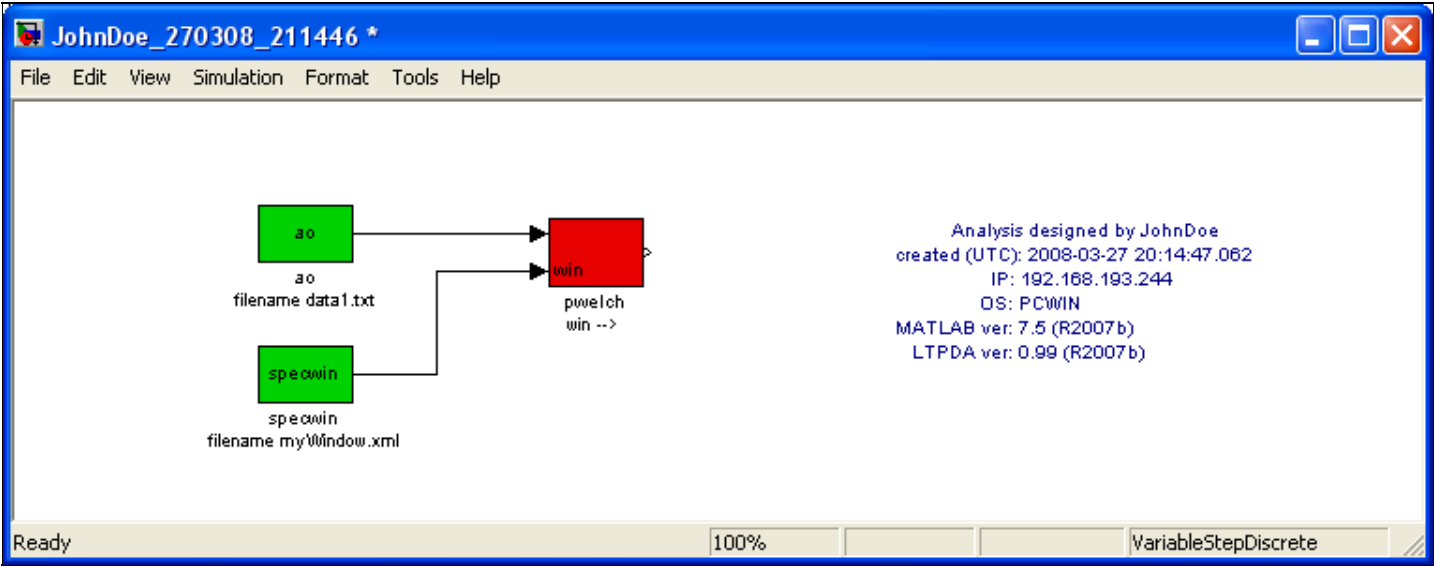
If the parameter is not supposed to be set in the GUI's panel but it must be retrieved from Simulink (for example, using a Constructor block for that particular type of object), the user can click on the

small 'From Simulink' checkbox.



The line will be updated, showing 'From Simulink environment'. Since this parameter will be retrieved from Simulink, the user cannot alter it anyway in the current parameters panel.

In the LTPDA Simulink model, the block will be modified with a new input, with the same name as the parameter's key:



While the first inport must be connected to the data coming in (if any is expected), as usual, the second port must be connected to the constructor block providing the object which will become the parameter.

In the example shown, the specwin Constructor block provide the specwin object that, inside the 'pwe1ch' block, will be retrieved as a parameter and passed to the function.

If the block had no data input, i.e. no input prior to the 'From Simulink' parameter, it will present a single inport, which must be connected to the Constructor block providing the parameter's contents.

Since it's possible to set multiple parameters coming from Simulink, the user should take care of the connections to the block, to avoid to shuffle the connected Constructor blocks – for example, inverting a specwin object and a MIIR object.

Load button

For MIIR and PZmodel parameters the GUI will show also a 'Load' button: the purpose of course is to retrieve these objects from a file on disk. Please note that for now this will convert the loaded object into its constructive plist, if possible, and the GUI will show again the parameters of the loaded object into the common parameters panel.

This will be discontinued or modified after R1

Adding parameters

Immediately below the last parameter, on the left, the GUI shows a popup list containing all the parameters types which can be added, and aside a '+' button to add a parameter of the selected type. Since it makes no sense to pass to a function 2 specwin object, PZmodels, MIIR filters, poles and zeros lists, these types won't be available in the popup list if any object of the same type is already shown in the parameters panel.

Reset parameter

In the lower left corner of the window the GUI draws the 'Reset parameters' button: this will reset the block to its empty initial state. If its a function/method block the GUI will then ask again for its list of required parameters, just like the block was just added to the model from the library.

Set name checkbox

At the bottom of the window, center line, there is the 'Set name' checkbox.

Enabling this the name of the output of the selected block will be assigned equal to the block's name (which can be altered freely by the user, standing the limitations on the block's name posed by Simulink); this will let the user to set particular and meaningful names instead of the automatically assigned ones, produced by the inner calculation functions.

The plot of an object assigned a particular name, for example, will show the proper name set.

Keep local result

Similarly to the 'Maintain intermediate results' checkbox in the main panel, this checkbox will mark the output of the selected block to be maintained at the end of the analysis, instead of being deleted just like any other intermediate result of the calculation.

In the LTPDA Simulink model the block will be shown with a purple background, to mark it as a 'probe' and to make it immediately recognizable.

Disabling the checkbox the block's background will be back to its common color, and the intermediate result produced by the block itself will be cleared at the end of the analysis.

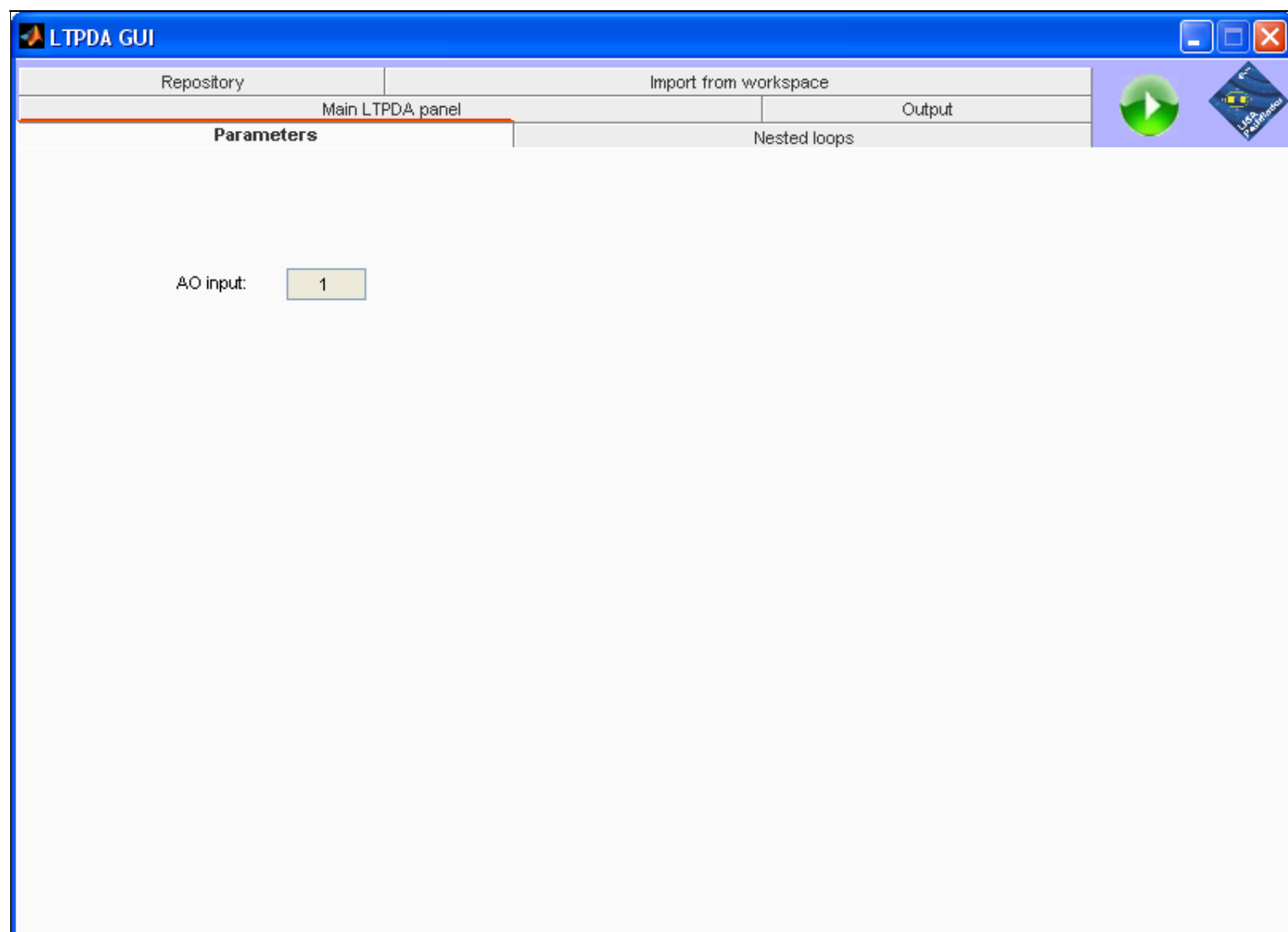
Help button

In the lower right corner of the window the GUI draws the '?' help button: this will recall the help associated to the function or method contained into the selected block, and it will show it into a new window.

The help shown is exactly the same available by the MATLAB command line, typing `help function_name`.

'Object from list' parameter

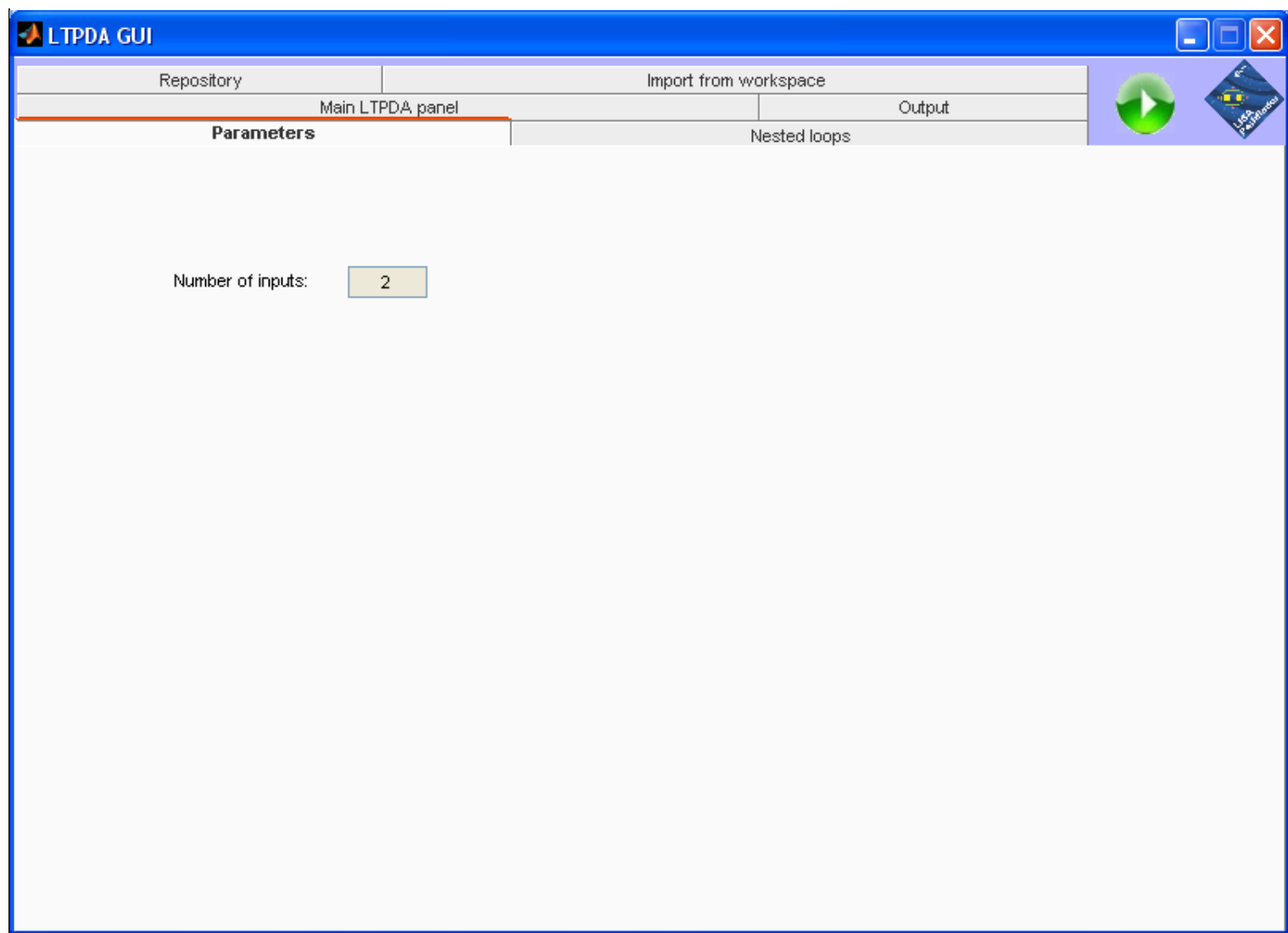
If the user selected a 'Object from list' (input) block into the active LTPDA Simulink model, the GUI will show:



The only parameter which can be set in this case is the reference index contained into the input block selected: the index will be used together with the Input array, so setting '1' will send to the connected block(s) the 1st element of the Input array.

Mux/Demux parameter

Similarly to the 'Input from list' block, the GUI will answer a selected a Mux block into a valid LTPDA Simulink model with:

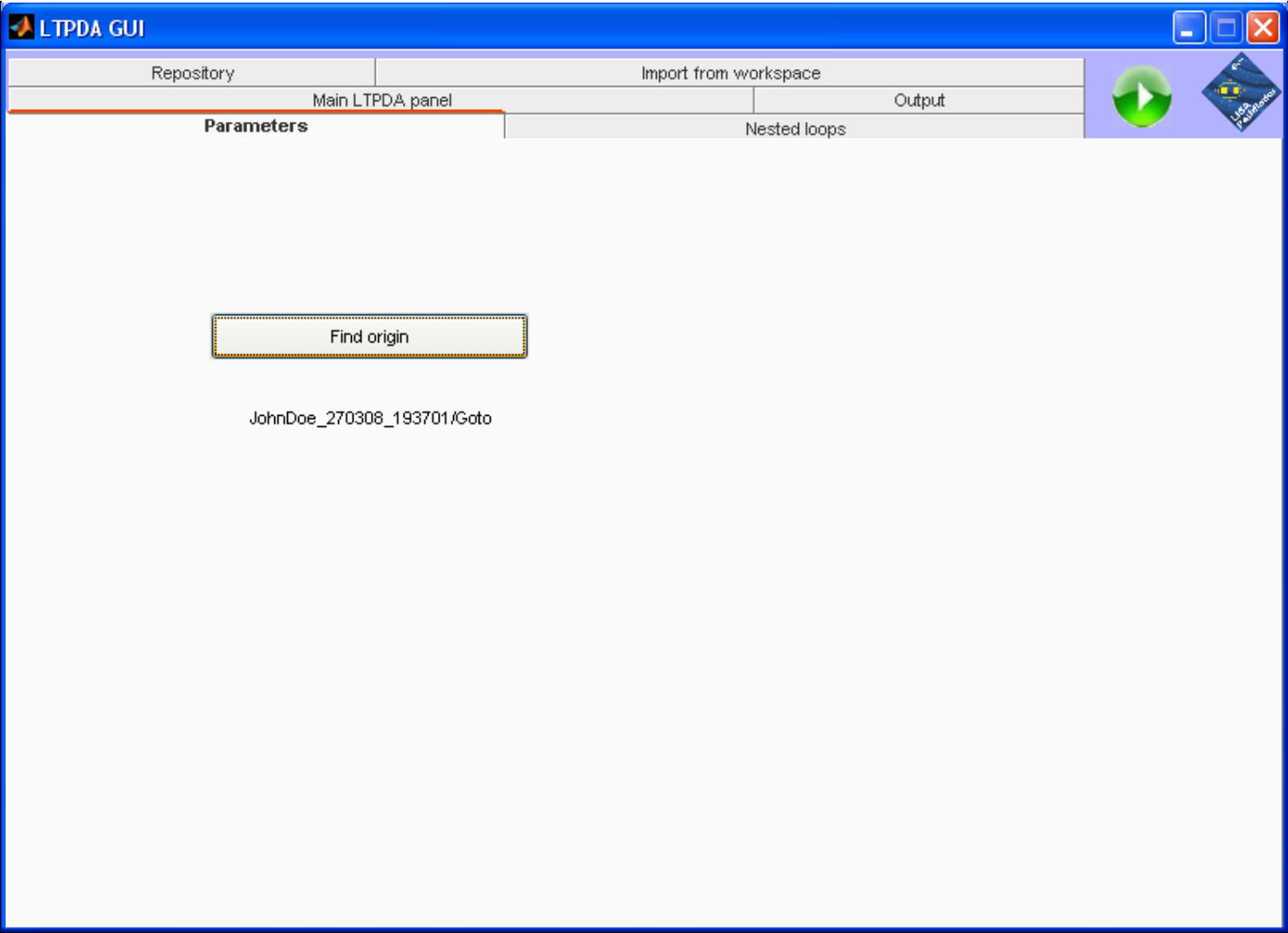


Changing this parameter will immediately alter the mux block in Simulink; unlike the direct setting of the number of inputs from Simulink, the GUI will change also the size of the block, in order to make it easier to connect the blocks and to be read.

The same applies to Demux blocks.

'From' block parameter

When the user selects a 'From' block into a valid LTPDA Simulink model, the GUI answers drawing the button 'Find origin', which will provide a direct pointing to the address of the 'Goto' block source of that signal.



If there's more than a single 'Goto' block, or if there's none, the GUI will show a proper warning.

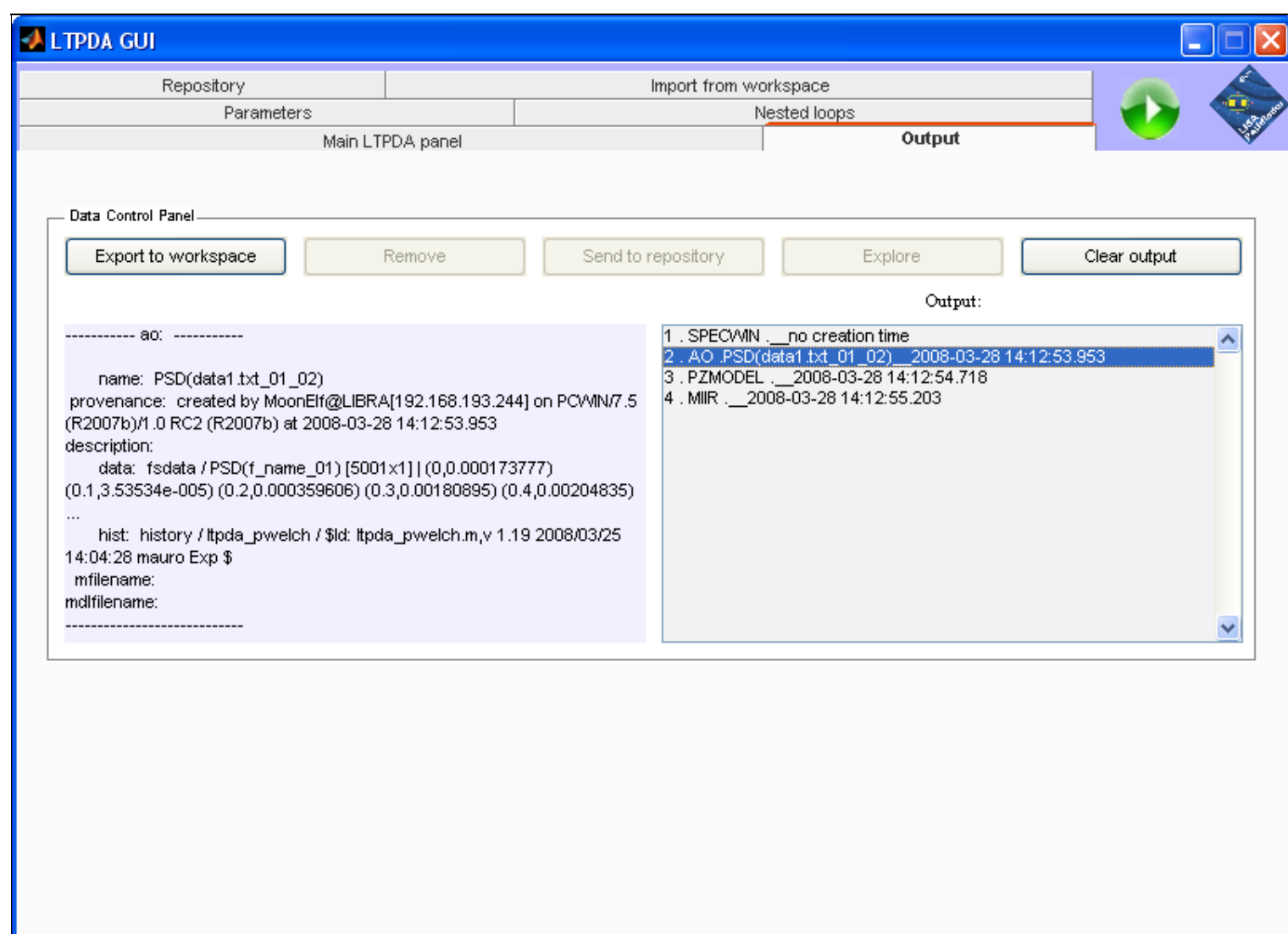
◀ Main panel

Output panel ▶

Output panel

The Output panel summarizes the contents of the Output array.

To save objects into the Output array, separating them from the other input objects created during the analysis, the user must use the proper 'Send to output' block in the Simulink model. The object received from the block will be just copied into the Output array.



Similarly to the Main panel, the central part of the window consists of 2 boxes:

- On the right, the list of objects contained in **the Output array**.
- On the left, **the info panel** showing the details of the selected object in the list aside.

While selecting an object from the list will show its details in the info panel on the left, similarly to what happens in the main panel, double clicking an object here will produce (if possible) a new window with its plot – instead of adding an 'Object from list' block to the model, as double clicking an object in the Input array list of the main panel.

In the upper part of the panel there are 5 buttons:

- **Export to workspace:** just like the same button in the main panel, this will save the selected object(s) into the MATLAB workspace. The name of these objects will be automatically set.
- **Remove:** *not for R1*
- **Send to repository:** *not for R1*
- **Explore:** *not for R1*
- **Clear output:** to clear the entire Output array.

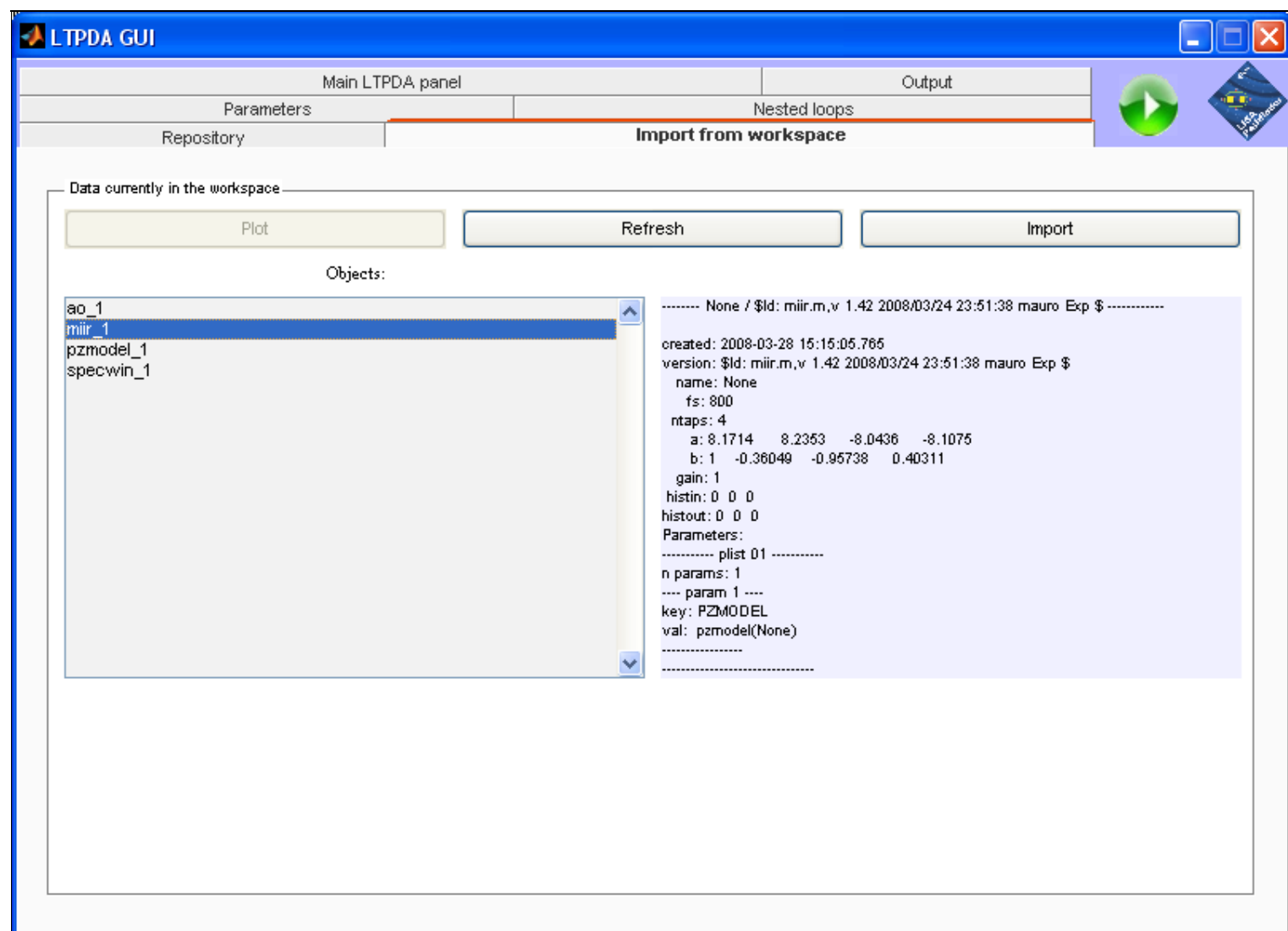
 Parameters

Import panel 

©LTP Team

Import panel

The Import panel lets the user to import into the LTPDA GUI workspace LTPDA objects currently residing in the main MATLAB workspace.



Similarly to the Main panel, the central part of the window consists of 2 boxes:

- On the left, the list of LTPDA objects currently stored in **the MATLAB workspace**. Please note that only LTPDA objects will be shown here.
- On the right, **the info panel** showing the details of the selected object in the list aside.

In the upper part of the panel there are 3 buttons:

- **Plot:** *not for R1*
- **Refresh:** to update the list of LTPDA objects in the MATLAB workspace, on the left.
- **Import:** to import the selected object(s) from the MATLAB workspace to the Input array. The object(s) will appear in the Input list of the main panel.

Repository panel

Not meant for R1

©LTP Team



Nested loops

Not meant for R1

Repository panel

The LTPDA Repository GUI

©LTP Team

The LTPDA Repository GUI

The LTPDA toolbox contains a client interface that can be used to interact with an LTPDA repository (see [Working with an LTPDA Repository](#)). The client interface can be accessed using the [LTPDA Repository GUI](#).

The pole/zero model helper

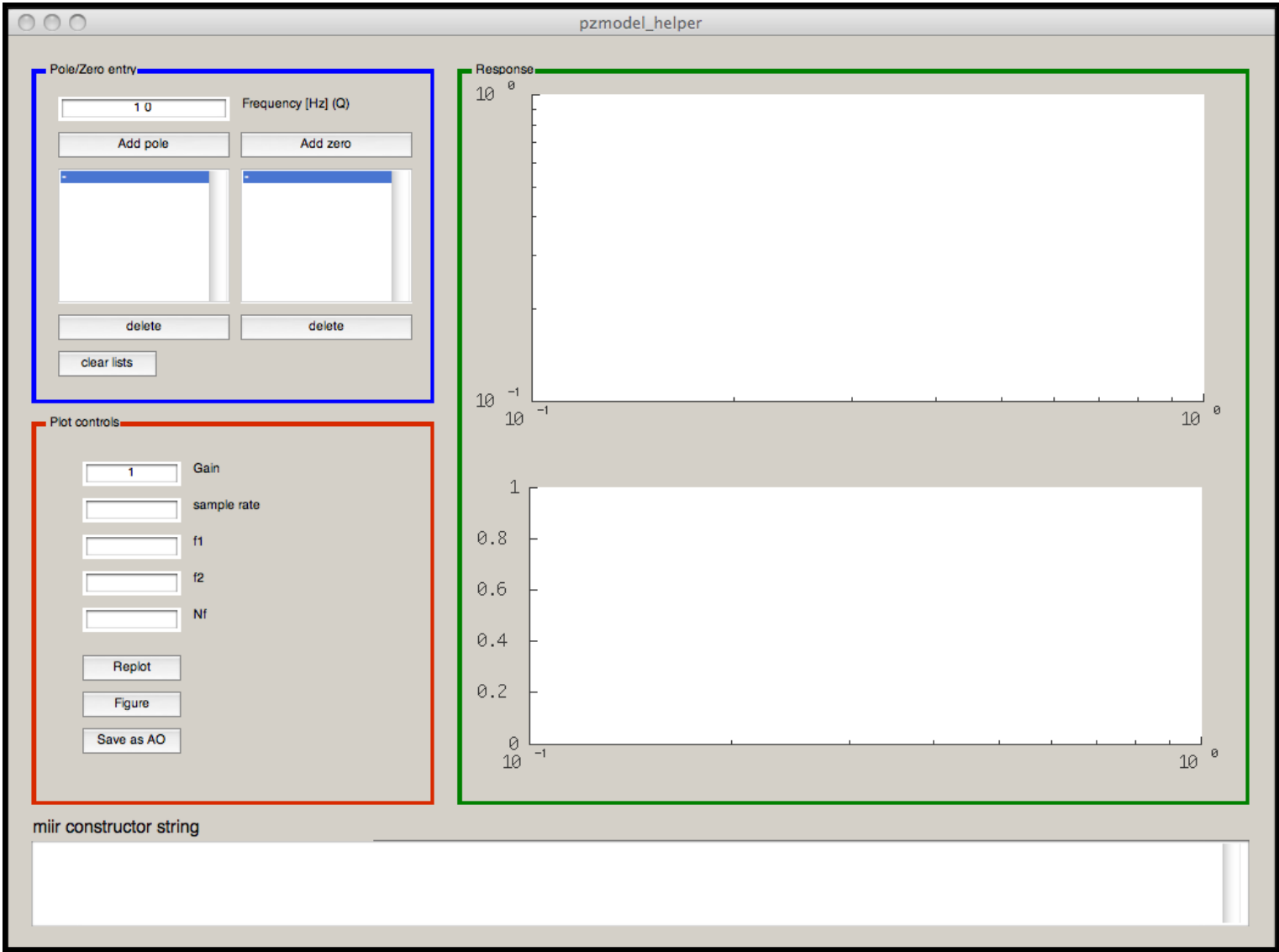
The LTPDA toolbox contains a class (`pzmodel`) for creating and using pole/zero models. The pole/zero model helper GUI allows the user to visualise the pole/zero model as it's being designed. It also allows the user to quickly see how the corresponding IIR filter (`miir` object) will look for different sample rates.

To start the pole/zero model helper:

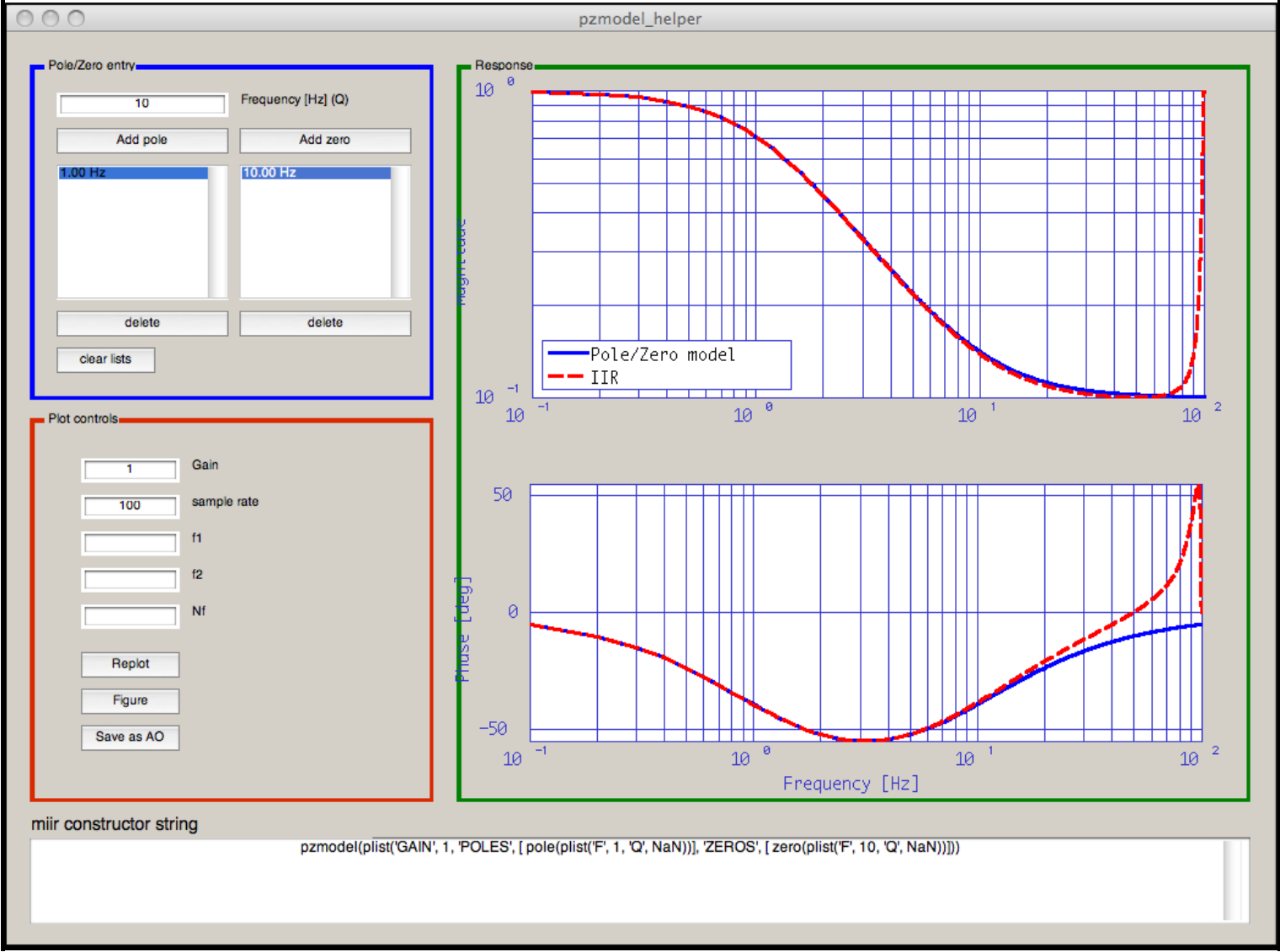
```
>> pzmodel_helper
```

or click the appropriate button on the LTPDA Launch Bay.

Once the GUI is loaded, you will see the following figure:



You can add poles and zeros to the model by entering the frequency (and Q) in the edit boxes, then click `add pole` or `add zero` as appropriate. The response is then updated in the response axes.



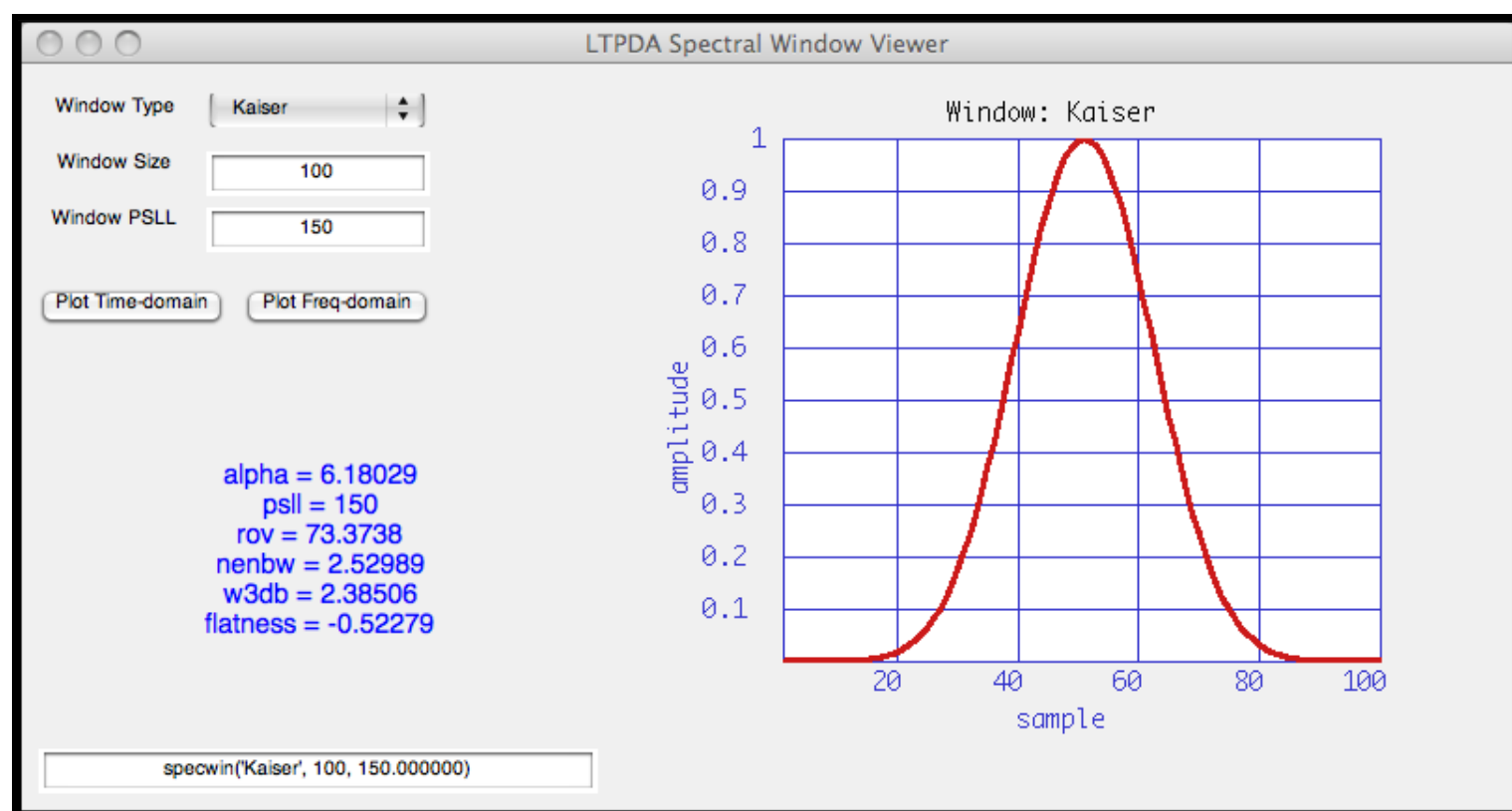
The Spectral Window GUI

The LTPDA Toolbox contains a class for creating spectral window objects (see [Spectral Windows](#)). A graphical user interface allows the user to easily explore the time-domain and frequency-domain response of any particular window.

To start the GUI:

```
>> specwin_viewer
```

or click the appropriate button on the Launch Bay.
You should then be presented with the following figure:



The constructor helper

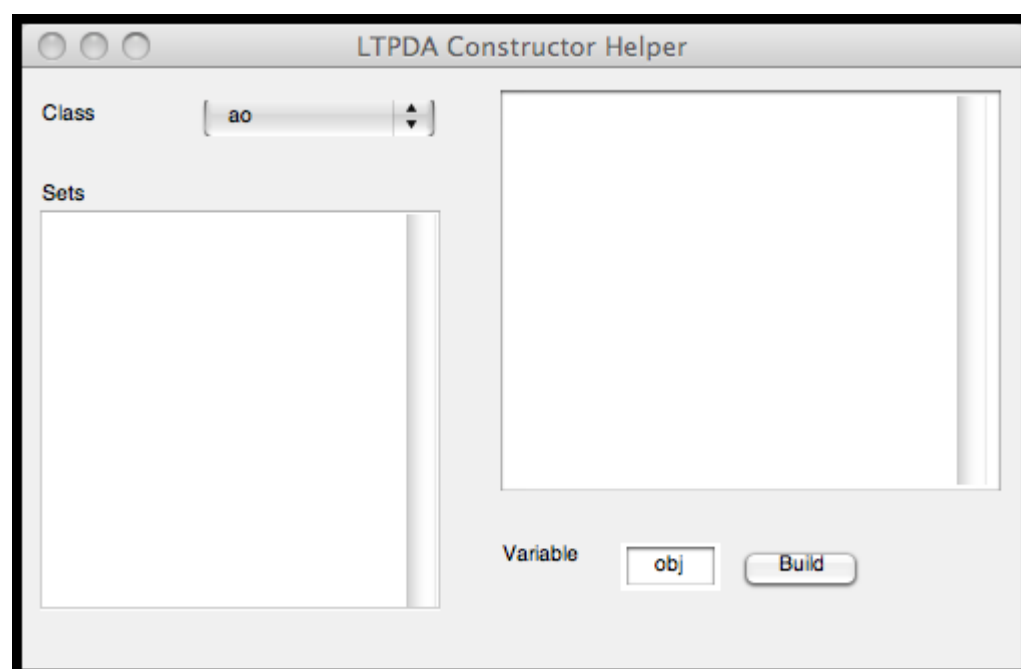
Since LTPDA is an object-oriented system, the user must create objects of different types using the appropriate constructors. The various constructor forms for each different LTPDA class can be explored using the constructor helper.

To start the constructor helper GUI:

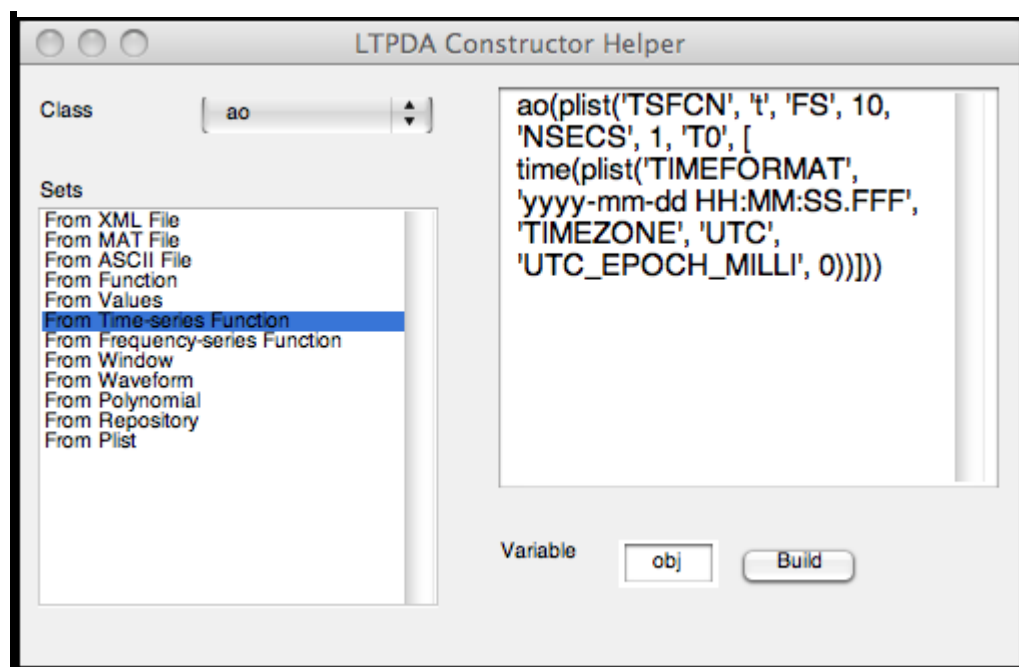
```
>> ltpda_constructor_helper
```

or click the appropriate button on the Launch Bay.

You should then be presented with the following figure:



Selecting a class from the drop-down list reveals the possible parameter sets for that class constructor. Selecting a parameter set reveals the default parameter list constructor string for constructing that class object in this way. For example, if we want to construct and Analysis Object using the time-series constructor, select the AO class then click on "From Time-series Function". You should then see:



You can then edit the parameter list and build the object by clicking on `Build`.

◀ The Spectral Window GUI

The LTPDA object explorer ▶

©LTP Team

The LTPDA object explorer

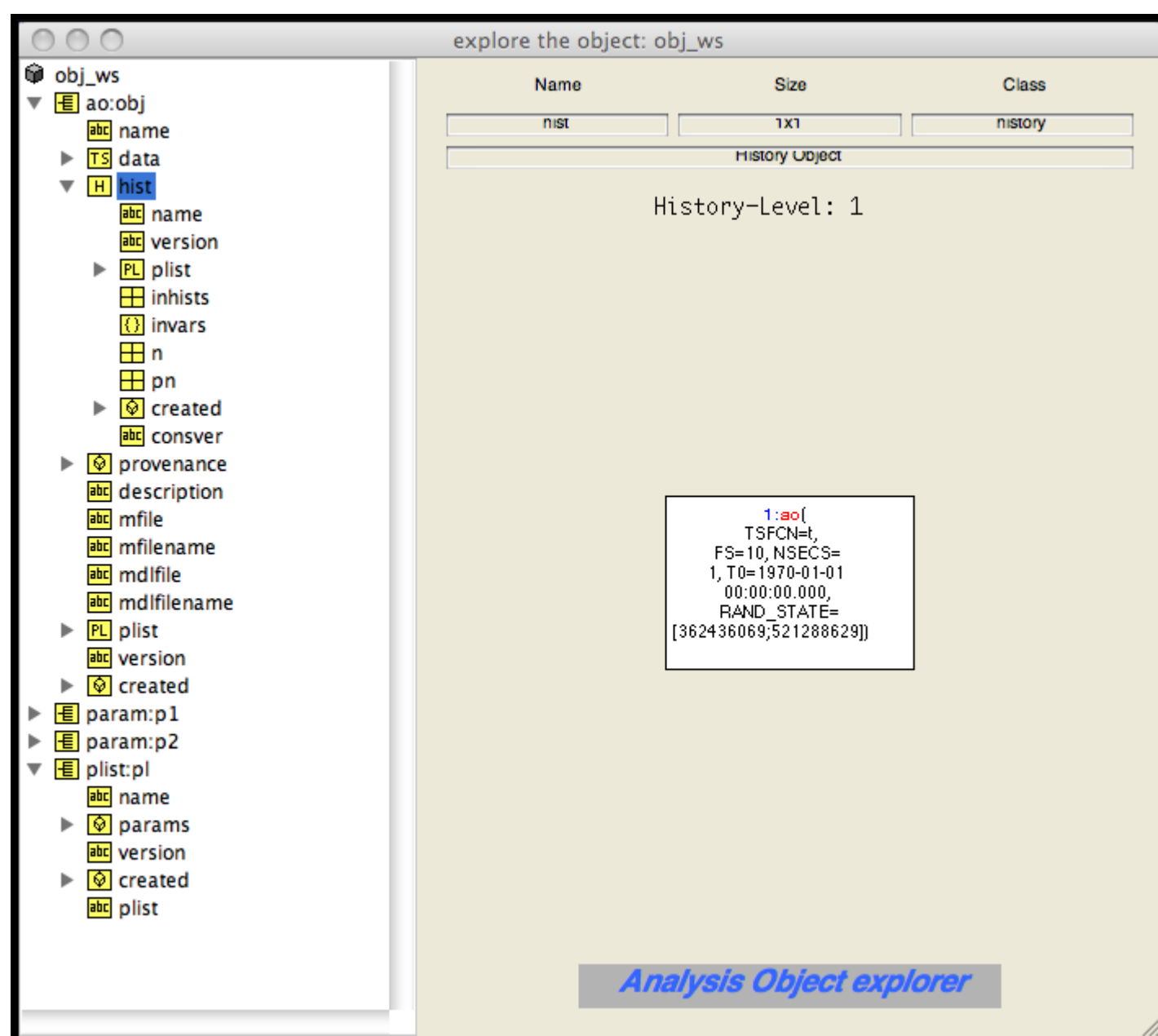
Since LTPDA works mainly with complex object types, it is often useful to explore the content of these objects graphically, particularly for Analysis Objects which may contain deep history trees. To do this, LTPDA offers the object explorer.

To start the object explorer:

```
>> explore_ao
```

or click the appropriate button on the Launch Bay.

The user is then presented with the following figure:



The object list is filled with all LTPDA User Objects currently in the MATLAB workspace.

◀ The constructor helper

The quicklook GUI ▶

©LTP Team

The quicklook GUI

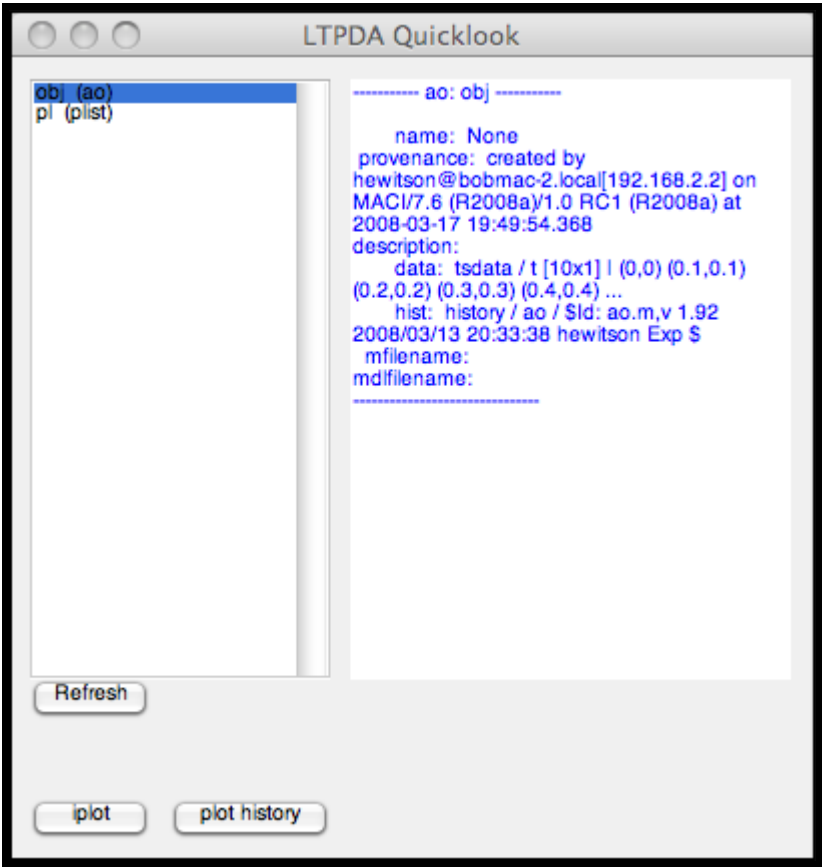
To quickly view all LTPDA objects currently in the MATLAB workspace, you can use the LTPDA Quicklook GUI.

To start the quicklook GUI:

```
>> ltpdaquicklook
```

or click the appropriate button on the Launch Bay.

The user will then be presented with the following figure:



The object list is filled with all LTPDA User Objects currently in the MATLAB workspace.

Working with an LTPDA Repository

Content needs written...

 The quicklook GUI	What is an LTPDA Repository 

©LTP Team

What is an LTPDA Repository

Introduction

An LTPDA repository has at its core a database server (in fact, a [MySQL server](#)). A single MySQL server can host multiple databases (LTPDA repositories). A single database/repository comprises a particular set of database tables. These tables hold meta-data about the objects stored in the database.

Since the core engine is a MySQL database, in principle any MySQL client can be used to interface with the repository. In order to submit and retrieve objects in the proper way (entering all expected meta-data), it is strongly suggested that you use the LTPDA Toolbox client commands `ltpda_obj_submit` and `ltpda_obj_retrieve` or the MATLAB [LTPDA repository GUI](#) (`repogui`).

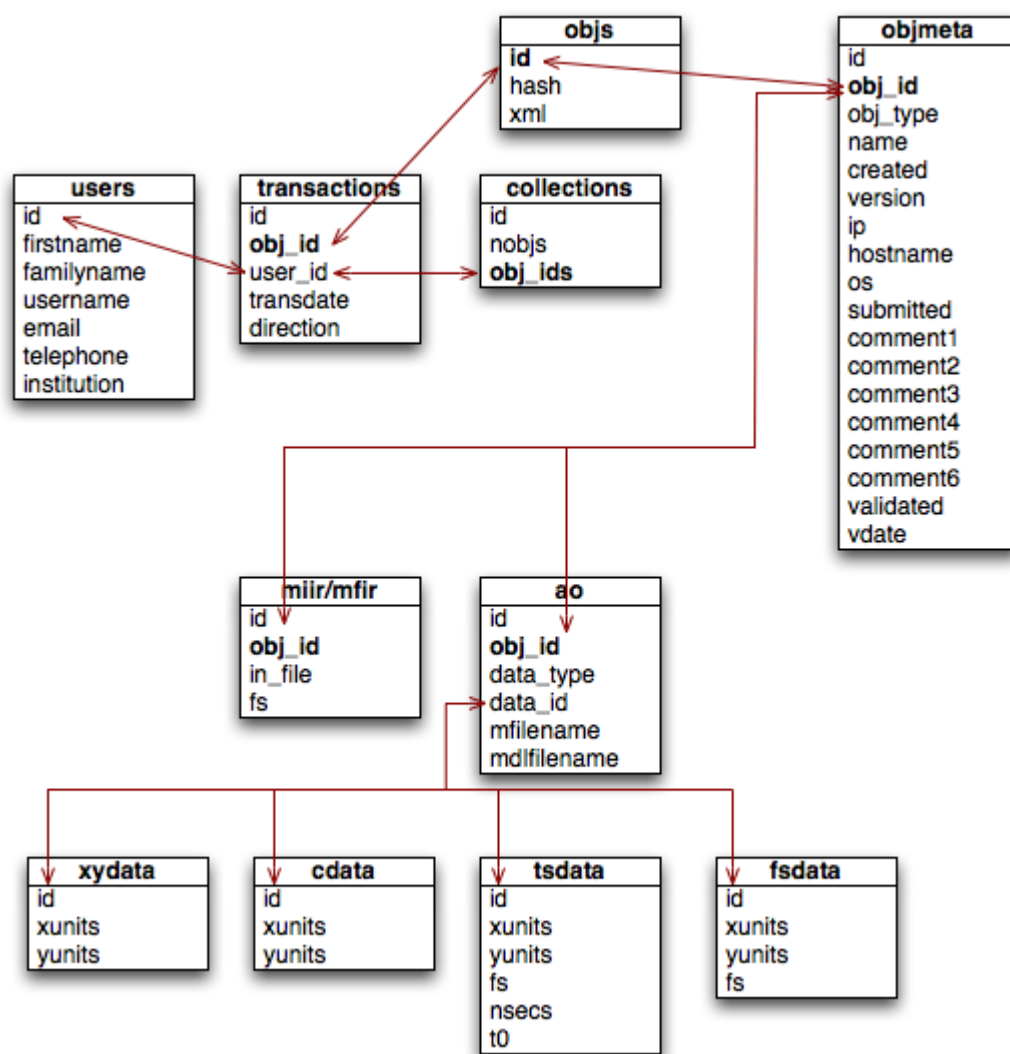
Any standard MySQL client can be used to query and search an LTPDA repository. For example, using a web-client or the standard MySQL command-line interface. In addition, the LTPDA Toolbox provides two ways to search the database: using the command `ltpda_dbquery` or using the LTPDA repository GUI. It is also possible to use the Visual Query Builder provided with the MATLAB Database Toolbox for interacting with a repository.

Database primer

A MySQL database comprises a collection of tables. Each table has a number of fields. Each field describes the type of data stored in that field (numerical, string, date, etc). When an entry is made in a table a new row is created. Interaction with MySQL databases is done using Standard Query Language (SQL) statements. For examples see [MySQL Common Queries](#).

Database design

The database for a single repository uses the tables as shown below:



As you can see, each object that is submitted to a repository receives a unique ID number. This ID number is used to link together the various pieces of meta-data that are collected about each object. In addition, each object that is submitted is check-summed using the [MD5 algorithm](#). That way, the integrity of each object can be checked upon retrieval.

In order to access a particular repository you need:

- The IP address of the MySQL host server
- The name of the repository (the database name)
- An account on the MySQL host server
- Permissions to access the desired database/repository

The main database tables

An LTPDA repository consists of the following database tables:

objs table

The objs table stores the XML representation of the submitted object. At this point, each object in the database is assigned a unique identifier. Together with the database name and hostname/ip of the server, this forms a unique tag for all LTPDA objects.

Field	Data Type	Description
id	int(11)	A unique identification number for all LTPDA objects in this database. This value is the link between all database tables.

hash	text	An MD5 hash of the XML representation of the object.
xml	longtext	The XML representation of the object. This field can be dumped directly to an XML file and should be readable in LTPDA.

objmeta table

The objmeta table stores various pieces of information associated with the object being submitted. The aim of this table is to provide a lot of useful fields on which to perform searches and queries.

Field	Data Type	Description
id	int(11)	A unique identification for all entries in this table.
obj_id	int(11)	The object id of the object in the objs table.
obj_type	text	The (LTPDA) class of this object.
name	text	The user-assigned name of this object.
created	datetime	The date and time this object was created.
version	text	The CVS tag of the object constructor code.
ip	text	The IP address of the machine which submitted the object.
hostname	text	The hostname of the machine which submitted the object.
os	text	The operating system of the machine which submitted the object.
submitted	datetime	The date and time the object was submitted.
experiment_title	text	A title for the experiment associated with the object.
experiment_desc	text	A description of the experiment associated with the object.
analysis_desc	text	A description of the analysis associated with the object.
quantity	text	If applicable, the physical quantity associated with the data in the object.
additional_authors	text	Any additional people involved in creating

		this object.
additional_comments	text	A free-form field of additional comments.
keywords	text	A list of keywords associated with the object.
reference_ids	text	ID numbers of any other objects associated with this object.
validated	tinyint(1)	A boolean field intended to indicate validated objects.
vdate	datetime	The date/time the object was validated.

transactions table

The transactions table records all user transactions. A transaction corresponds to submitting or retrieving a single or a collection of LTPDA objects.

Field	Data Type	Description
id	int(11)	A unique identification number this table entry.
obj_id	int(11)	The object id of the object in the objs table.
user_id	int(11)	The unique ID number of the user who carried out the transaction.
transdate	datetime	The date/time of the transaction.
direction	text	The direction of the transaction: 'in' or 'out'.

users table

The users table stores information about the users allowed to access the database.

Field	Data Type	Description
id	int(11)	A unique identification number for all entries in this table.
firstname	text	The firstname of the user.
familyname	text	The family name of the user.
username	text	The username (login name) of the user.
email	text	A valid e-mail address for the user.
telephone	text	A telephone numnber for the user.

institution	text	The institution of the user.

collections table

The `collections` table stores virtual collections of objects submitted to the database. When the user submits one or more objects at the same time, this constitutes a collection. In this case a collection ID number is assigned next to a list of the object IDs in the collection. This allows the user to retrieve collections of objects based on the collection ID alone; no information about the individual object IDs is required.

Field	Data Type	Description
id	int(11)	A unique identification number for this collection of objects.
nobjs	int	The number of objects in the collection.
obj_ids	text	A comma separated list of object IDs.

Additional database tables

As well as the main database tables, additional meta-data tables are used to capture extra meta-data about some of the LTPDA objects.

ao table

The `ao` table stores additional meta-data specific to analysis objects.

Field	Data Type	Description
id	int(11)	A unique identification number for all entries in the table.
obj_id	int(11)	The unique ID of the object.
data_type	text	The type/class of the data stored in the AO.
data_id	int(11)	The unique ID of the data object listed in one of the data meta-data tables.
description	text	The description property of the AO.
mfilename	text	The filename of any m-file attached to the AO.
mdlfilename	text	The filename of any SIMULINK model file attached to the AO.

miir table

The `miir` table stores additional meta-data specific to miir filter objects.

Field	Data Type	Description
id	int(11)	A unique identification number for all entries in the table.
obj_id	int(11)	The unique ID of the object.
in_file	text	The input filename (if applicable) used to create the filter object
fs	int(11)	The sample rate of the data the filter is designed for.

mfir table

The mfir table stores additional meta-data specific to mfir filter objects.

Field	Data Type	Description
id	int(11)	A unique identification number for all entries in the table.
obj_id	int(11)	The unique ID of the object.
in_file	text	The input filename (if applicable) used to create the filter object
fs	int(11)	The sample rate of the data the filter is designed for.

tsdata table

The tsdata table stores additional meta-data specific to tsdata (time-series data) objects.

Field	Data Type	Description
id	int(11)	A unique identification number for all entries in the table.
xunits	text	The X-units associated with this time-series.
yunits	text	The Y-units associated with this time-series.
fs	int(11)	The sample rate of the data the filter is designed for.
nsecs	int	The duration (number of seconds) of data in the object.
t0	datetime	The date/time associated with the start (first sample) of the time-series.

fsdata table

The fsdata table stores additional meta-data specific to fsdata (frequency-series data) objects.

Field	Data Type	Description

id	int(11)	A unique identification number for all entries in the table.
xunits	text	The X-units associated with this time-series.
yunits	text	The Y-units associated with this time-series.
fs	int(11)	The sample rate of the data the filter is designed for.

cdata table

The cdata table stores additional meta-data specific to cdata (1D data) objects.

Field	Data Type	Description
id	int(11)	A unique identification number for all entries in the table.
xunits	text	The X-units associated with this time-series.
yunits	text	The Y-units associated with this time-series.

xydata table

The xydata table stores additional meta-data specific to xydata (2D data) objects.

Field	Data Type	Description
id	int(11)	A unique identification number for all entries in the table.
xunits	text	The X-units associated with this time-series.
yunits	text	The Y-units associated with this time-series.

Connecting to an LTPDA Repository

Connection to an LTPDA Repository uses the JDBC interface of the Database toolbox. The command `mysql_connect` can be used to connect to a repository. It takes the following input arguments:

hostname	A hostname for the repository
dbname	A database name to connect to

You will then be prompted for a valid username and password. Here is an example call:

```
>> conn = mysql_connect('localhost', 'ltpda_test')
** Connecting to localhost as ltpdaadmin...
** Connection status:
    DatabaseProductName: 'MySQL'
    DatabaseProductVersion: '5.0.45'
    JDBCDriverName: 'MySQL-AB JDBC Driver'
    JDBCDriverVersion: [1x103 char]
    MaxDatabaseConnections: 0
    CurrentUserName: 'ltpdaadmin@localhost'
    DatabaseURL: 'jdbc:mysql://localhost/ltpda_test'
    AutoCommitTransactions: 'True'
```

The result is a database object which can be further used to interact with the repository. To disconnect from the server, use the `close` method of the database class:

```
>> close(conn)
```


Submitting LTPDA objects to a repository

Any of the following user objects can be submitted to an LTPDA repository:

- ao
- miir
- mfir
- pzmodel
- timespan
- ssm
- plist

The submission process

When an object is submitted, the following steps are taken:

1. The `userid` of the user connecting is retrieved from the `Users` table of the repository
2. For each object to be submitted:
 1. The object to be submitted is checked to be one of the types listed above
 2. The `name`, `created`, and `version` fields are read from the object
 3. The object is converted to an XML text string
 4. An MD5 hash sum is computed for the XML string
 5. The XML string and the hash code are inserted in to the `objs` table
 6. The automatically assigned ID of the object is retrieved from the `objs` table
 7. Various pieces of meta-data (object name, object type, created time, client IP address, etc.) are submitted to the `objmeta` table
 8. Additional meta-data is entered into the table matching the object class (`ao`, `tsdata`, etc.)
 9. An 'in' entry is made in the `transaction` table recording the user ID and the object ID
3. A entry is then made in the `collections` table, even if this is a single object submission
4. The object IDs and the collection ID are returned to the user

Submitting objects

Objects can be submitted using the command `submit`. This command takes at least two inputs:

object	The LTPDA object to submit
sinfo	An information structure (see below)

The information structure should have the following fields:

'conn'	- database connection object
'experiment_title'	- a title for the submission (Mandatory, >4 characters)
'experiment_description'	- a description of this submission (Mandatory, >10 characters)
'analysis_description'	- a description of the analysis performed (Mandatory, >10 characters));
'quantity'	- the physical quantity represented by the data);
'keywords'	- a comma-delimited list of keywords);

```
'reference_ids'      - a string containing any reference object id numbers
'additional_comments' - any additional comments
'additional_authors' - any additional author names
```

The following example script connects to a repository and submits an AO:

```
% Connect to a repository
conn = utils.mysql.connect('localhost', 'ltpda_test');

% Load the AO
a = ao('result.xml');

% Build an information structure
sinfo.conn          = conn;
sinfo.experiment_title = 'Interferometer noise';
sinfo.experiment_description = 'Spectral estimation of interferometer output signal';
sinfo.analysis_description = 'Spectrum of the recorded signal';
sinfo.quantity       = 'photodiode output';
sinfo.keywords        = 'interferometer, noise, spectrum';
sinfo.reference_ids   = '';
sinfo.additional_comments = 'none';
sinfo.additional_authors = 'no one';

% Submit the AO
[ids, cid] = submit(a, sinfo);

% Close the connection
close(conn);
```

Submitting collections

Collections of LTPDA objects can also be submitted. Here a collection is defined as a group of objects submitted at the same time. In this way, a single information structure describing the collection is assigned to all the objects. The collection is just a virtual object; it is defined by a list of object IDs in the database. The following example script connects to a repository and submits three AOs:

```
% Connect to a repository
conn = utils.mysql.connect('localhost', 'ltpda_test');

% Create objects to submit
o1 = ao(plist('waveform', 'sine wave', 'f', 1, 'phi', 0, 'nsecs', 10, 'fs', 100));
o2 = pzmodel(1, 10, 100);
o3 = plist('b', 2, 'c', 'asd');

% Create an information structure
sinfo.conn          = conn;
sinfo.experiment_title = 'submit multiple objects';
sinfo.experiment_description = 'this is just a test of the whole thing';
sinfo.analysis_description = 'no analysis this time';
sinfo.quantity       = '';
sinfo.keywords        = '';
sinfo.reference_ids   = '';
sinfo.additional_comments = 'none';
sinfo.additional_authors = 'no one';

% Submit the objects
[ids, cid] = submit(o1, o2, o3, sinfo);

% Close connection
close(conn);

% END
```

Running this script yields the following output:

```
** Connecting to 130.75.117.67 as hewitson...
** Connection status:
```

```

DatabaseProductName: 'MySQL'
DatabaseProductVersion: '5.0.45'
JDBCDriverName: 'MySQL-AB JDBC Driver'
JDBCDriverVersion: [1x56 char]
MaxDatabaseConnections: 0
CurrentUserName: 'hewitson@pixfirewall.aei.uni-hannover.de'
DatabaseURL: 'jdbc:mysql://130.75.117.67/ltpda_test'
AutoCommitTransactions: 'True'

M: running ao/ao
M: constructing from plist
M: running ltpda_uo/submit
M: sinfo structure is valid.
M: submitting objects to repository.
M: got user id 2 for user: hewitson
M: submitting object: ao / sine wave
M: uploading XML data...
M: done.
M: submitted object ao with id 1624
M: running query INSERT INTO objmeta SET obj_id=1624,obj_type='ao',name='sine
wave',created='2008-09-02 17:25:52',version='$Id: repo_submit_content.html,v 1.4 2008/09/02
17:56:59 hewitson Exp $',ip='192.168.2.100',hostname='martin-hewitsons-macbook-
pro.local',os='MACI',submitted='2008-09-02 19:25:53',experiment_title='submit multiple
objects',experiment_desc='this is just a test of the whole
thing',reference_ids='',additional_comments='none',additional_authors='no
one',keywords='',quantity='',analysis_desc='no analysis this time';
M: made meta-data entry
M: running query INSERT INTO tsdata SET xunits='s',yunits='V',fs=100,nsecs=10,t0='1970-01-
01 00:00:00';
M: running query INSERT INTO ao SET
obj_id=1624,data_type='tsdata',data_id=1263,description='',mfilename='',mdlfilename='';
M: running query INSERT INTO transactions SET obj_id=1624,user_id=2,transdate='2008-09-02
17:25:52',direction='in';
M: updated transactions table
M: submitting object: pzmodel / None
M: uploading XML data...
M: done.
M: submitted object pzmodel with id 1625
M: running query INSERT INTO objmeta SET
obj_id=1625,obj_type='pzmodel',name='None',created='2008-09-02 17:25:52',version='$Id:
repo_submit_content.html,v 1.4 2008/09/02 17:56:59 hewitson Exp
$',ip='192.168.2.100',hostname='martin-hewitsons-macbook-
pro.local',os='MACI',submitted='2008-09-02 19:25:54',experiment_title='submit multiple
objects',experiment_desc='this is just a test of the whole
thing',reference_ids='',additional_comments='none',additional_authors='no
one',keywords='',quantity='',analysis_desc='no analysis this time';
M: made meta-data entry
M: running query INSERT INTO transactions SET obj_id=1625,user_id=2,transdate='2008-09-02
17:25:52',direction='in';
M: updated transactions table
M: submitting object: plist / none
M: uploading XML data...
M: done.
M: submitted object plist with id 1626
M: running query INSERT INTO objmeta SET
obj_id=1626,obj_type='plist',name='none',created='2008-09-02 17:25:52',version='$Id:
repo_submit_content.html,v 1.4 2008/09/02 17:56:59 hewitson Exp
$',ip='192.168.2.100',hostname='martin-hewitsons-macbook-
pro.local',os='MACI',submitted='2008-09-02 19:25:54',experiment_title='submit multiple
objects',experiment_desc='this is just a test of the whole
thing',reference_ids='',additional_comments='none',additional_authors='no
one',keywords='',quantity='',analysis_desc='no analysis this time';
M: made meta-data entry
M: running query INSERT INTO transactions SET obj_id=1626,user_id=2,transdate='2008-09-02
17:25:52',direction='in';
M: updated transactions table
M: running query INSERT INTO collections SET nobjs=3,obj_ids='1624,1625,1626';
M: made collection entry
M: submission complete.

```

Exploring an LTPDA Repository

Since an LTPDA repository is just a MySQL database, you can query the database using standard SQL commands via any of the popular MySQL clients. In addition, the LTPDA toolbox provides a simplified command that can be used to execute simple queries with only basic SQL knowledge.

The command is called `ltpda_dbquery` and it can be used to perform various queries. It takes the following input arguments:

conn	A database connection object
tablename	The name of a table to search
query	The query string written in MySQL SQL syntax

Examples of usage are:

Searching particular tables

```
>> info = ltpda_dbquery(conn, 'select * from objmeta where id>1000 and id<2000');
>> info = ltpda_dbquery(conn, 'ao', 'id>1000 and id<2000');
>> info = ltpda_dbquery(conn, 'objmeta', 'name like "x12"');
>> info = ltpda_dbquery(conn, 'users', 'username="aouser"');
>> info = ltpda_dbquery(conn, 'collections', 'id=3');
>> info = ltpda_dbquery(conn, 'collections', 'obj_ids="1,2"');
>> info = ltpda_dbquery(conn, 'transactions', 'user_id=3');
>> info = ltpda_dbquery(conn, 'transactions', 'obj_id=56');
```

Retrieving a list of tables

You can retrieve a list of the tables in a database with the call:

```
>> info = ltpda_dbquery(conn)
```

High-level queries

Various standard queries are envisaged which ask typical questions, such as: "Give me data for a particular signal spanning a particular time-span".

Formulating this question as an SQL query requires a good knowledge of the SQL syntax used by MySQL. The query has to search across multiple tables in order to gather the IDs of the objects that fulfill the query. For these standard questions, high-level functions will be built which perform the query given some input information. This avoids the user having to formulate complicated SQL statements.

The following high-level queries currently exist in the toolbox:

<code>ltpda_getAOsInTimeSpan</code>	Retrieve particular AOs in the given time-span

Retrieving LTPDA objects from a repository

Objects can be retrieved from the repository either by specifying an object ID or a collection ID. The LTPDA Toolbox provides the function `ltpda_uo.retrieve` to retrieve objects. In addition, the constructors of each user class can be used to retrieve objects of that class.

The retrieval process

When an object is retrieved, the following steps are taken:

1. The object type for the requested ID is retrieved from the `objmeta` table
2. A call is made to the appropriate class constructor
3. The class constructor retrieves the XML string from the `objs` table
4. The XML string is then converted into an XML Xdoc object
5. The Xdoc object is then parsed to recreate the desired object

Retrieving objects

To retrieve an object, you must know its object ID, or the ID of the collection that contains that object. The following script shows an example of retrieving a single object:

```
% Connect to a repository
[conn, username] = utils.mysql.connect('130.75.117.67', 'ltpda_test');

% Retrieve the object
q = ltpda_uo.retrieve(conn, 12);

% Close connection
close(conn);
```

If you already know the class of the object (for example, `ao`), you can do

```
% Retrieve the object
q = ao(plist('hostname', hostname, 'database', dbname, 'ID', 12));
```

If you know the collection that contains the object, and the class, then you can do

```
% Retrieve the object
q = ao(plist('hostname', hostname, 'database', dbname, 'CID', 2));
```

In this case, all AOs in the collection with ID 2 will be retrieved and stored in `q`.

Multiple objects can be retrieved simultaneously by giving a list of object IDs. For example

```
q = ltpda_uo.retrieve(conn, 1,2,3);
```

When multiple objects are requested, the results are returned in a cell array.

Retrieving object collections

Collections of objects can be retrieved by specifying the collection ID. The following script retrieves a collection:

```
% Connect to a repository
[conn, username] = utils.mysql.connect('130.75.117.67', 'ltpda_test');

% Retrieve the collection
q = ltpda_uo.retrieve(conn, 'Collection', 1);

% Close connection
close(conn);
```

The output is a cell array containing the objects retrieved.

◀ Exploring an LTPDA Repository

Using the LTPDA Repository GUI ▶

©LTP Team

Using the LTPDA Repository GUI

The LTPDA Toolbox provides a graphical user interface for interacting with an LTPDA repository.

- [Starting the repository GUI](#)
- [Connecting to a repository](#)
- [Submitting objects to a repository](#)
- [Querying the contents of a repository](#)
- [Retrieving objects and collections from a repository](#)

Starting the LTPDA Repository GUI

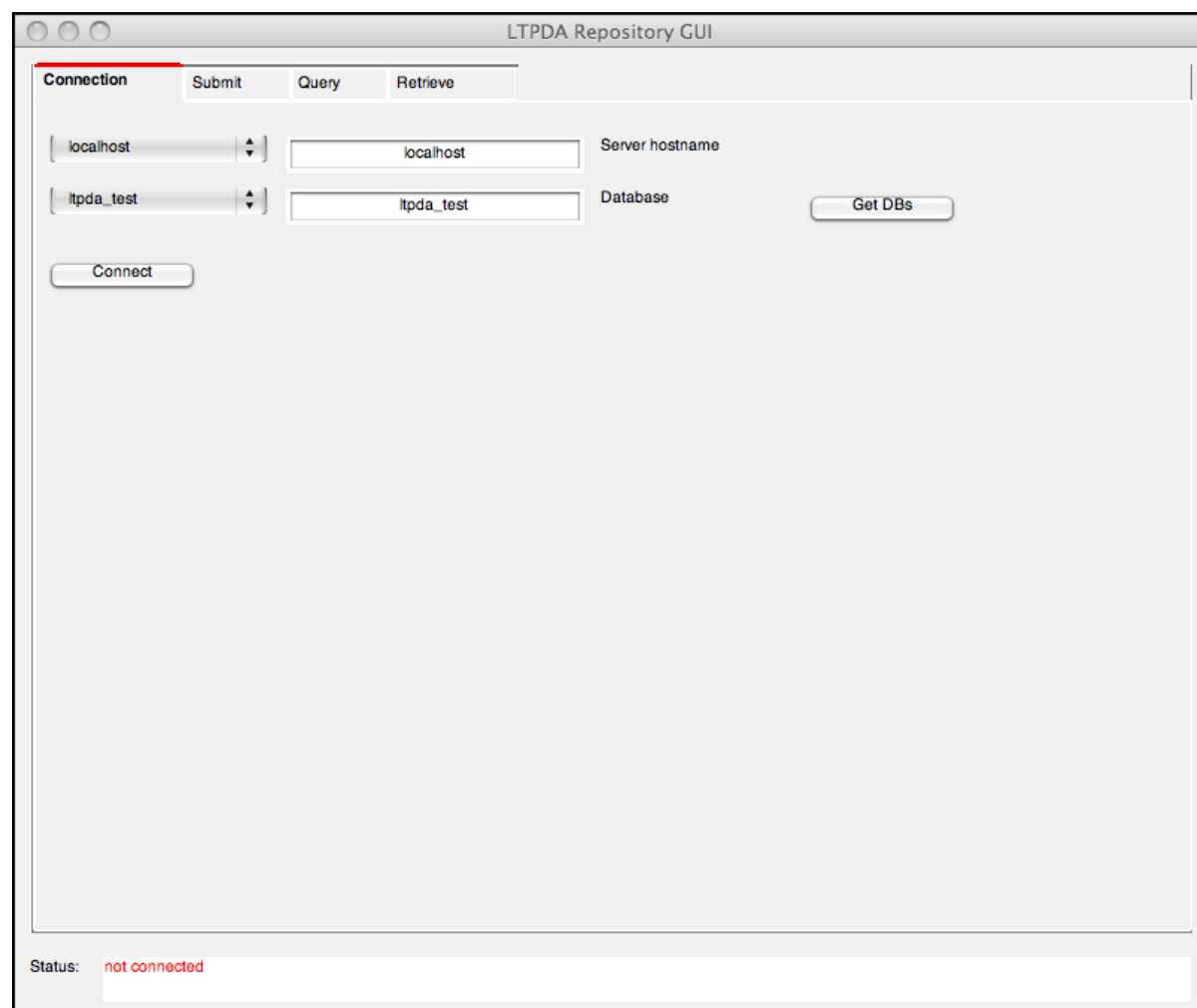
The GUI can be started using the command

```
>> repogui
```

The interface allows submission and retrieval of objects, as well as querying of a repository.

Connecting to a repository

The first tab pane is the connection panel.



The user can select one of the pre-defined hosts or type a new hostname or IP address into the Server hostname field. If the database name is already known, it can be entered directly in the text field. If the database name is not known, a list of LTPDA repositories available on that particular host can be retrieved by clicking on the 'Get DBs' button. If the user has not already authenticated with the host, a login dialog will be presented prompting the user for a username and password. If authentication is successful, the drop-down menu to the left of the database name text entry field will be filled with the names of the available repositories.

The user can then select a repository and connect to it by clicking the 'Connect' button. If the user has permissions to connect to that particular repository, the repository GUI will hold a database connection object for use on the other panels of the GUI.

Disconnecting from a repository

To disconnect from the current repository, click the 'Disconnect' button. This must be done before being able to connect to a different repository.

◀ Using the LTPDA Repository GUI

Submitting objects to a repository ▶

©LTP Team

Submitting objects to a repository

Objects can be submitted to the repository using the 'Submit' panel shown below.

The screenshot shows the LTPDA Repository GUI with the 'Submit' tab selected. On the left, a list of LTPDA Objects is displayed, including a1 (ao), a2 (ao), a3 (ao), a4 (ao), a5 (ao), a6 (ao), firf (mfir), iirf (miir), pl1 (plist), pzm (pzmmodel), s (specwin), t (time), and ts (timespan). A 'Refresh list' button is located below the list. On the right, there are several input fields for submission details, each with a red label indicating it is mandatory: 'Experiment Title' (filled with 'My Experiment'), 'Experiment description' (filled with 'My experiment was to do this and that with some data.'), 'Analysis description' (filled with 'The data was analysed using the magical new FOO algorithm.'), 'Quantity' (filled with 'Sensor output'), 'Keywords' (filled with 'FOO, sensor'), 'Reference IDs' (filled with 'none'), 'Additional Comments' (empty), and 'Additional Authors' (empty). A 'Submit' button is located at the bottom left of the form. The status bar at the bottom indicates 'Status: not connected'.

Selecting the objects to submit

The objects available for submission are LTPDA objects currently in the MATLAB workspace. Clicking the 'Refresh list' button will refresh the list of objects. The user can `ctrl-click` to select a subset of objects in the list for submission.

Completing the submission form

In order to ease subsequent usage of data submitted, and to allow for high level queries to be performed, the submission process must be completed with additional and sensitive informations associated with the data included in the objects. The first three fields, marked in red, are mandatory (a submission without specifying those informations will fail). The available fields are:

- **Experiment title** A short title for the experiment from which the data originate. Mandatory field, to be filled with more than 4 characters.
- **Experiment description** A description of the experiment from which the data originate.

Mandatory field, to be filled with more that 10 characters.

- **Analysis description** A description of the analysis performed on the data. **Mandatory field**, to be filled with more that 10 characters.
- **Quantity** The physical quantity that the data represent.
- **Keywords** A comma-delimited list of keywords.
- **Reference IDs** A list od object IDs that are relevant to this/these results.
- **Additional Comments** Anything else the user wants to say about the objects being submitted.
- **Additional Authors** A list of people who helped creating these object(s)

After inserting the useful information by filling the corresponding entries, the user can proceed with the submission by clicking the 'Submit' button. The Matlab window will show the response from the repository, including the IDs assigned to the submitted objects.

◀ Connecting to a repository

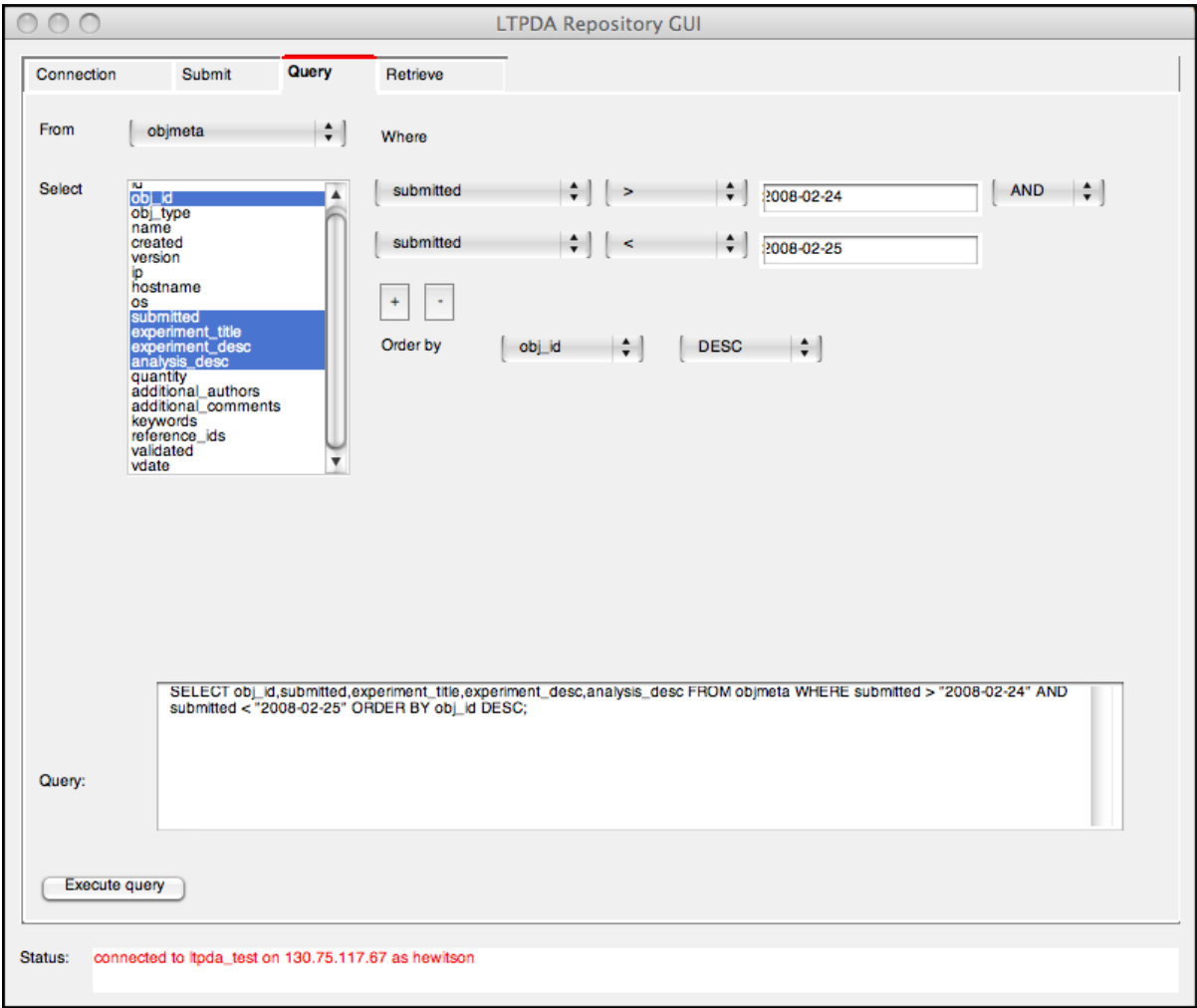
Querying the contents of a repository ▶

©LTP Team

Querying the contents of a repository

Querying an LTPDA repository is done using standard SQL statements. The repository GUI presents the user with the possibility to graphically build SQL statements which avoids learning SQL syntax. Currently, the SQL statements that can be built in this way are restricted to queries on a single database table. No high-level queries are currently implemented.

The repository GUI has a query panel which looks like the figure below:



In this figure, you see that the user has built a query to select all objects submitted on the 24th February 2008. The results will contain the object id, the submitted date, the experiment title, the experiment description, and the analysis description. The results will be sorted in descending order of the object id.

Executing the query (click the 'Execute query' button) produces the results table shown below.

Query Results

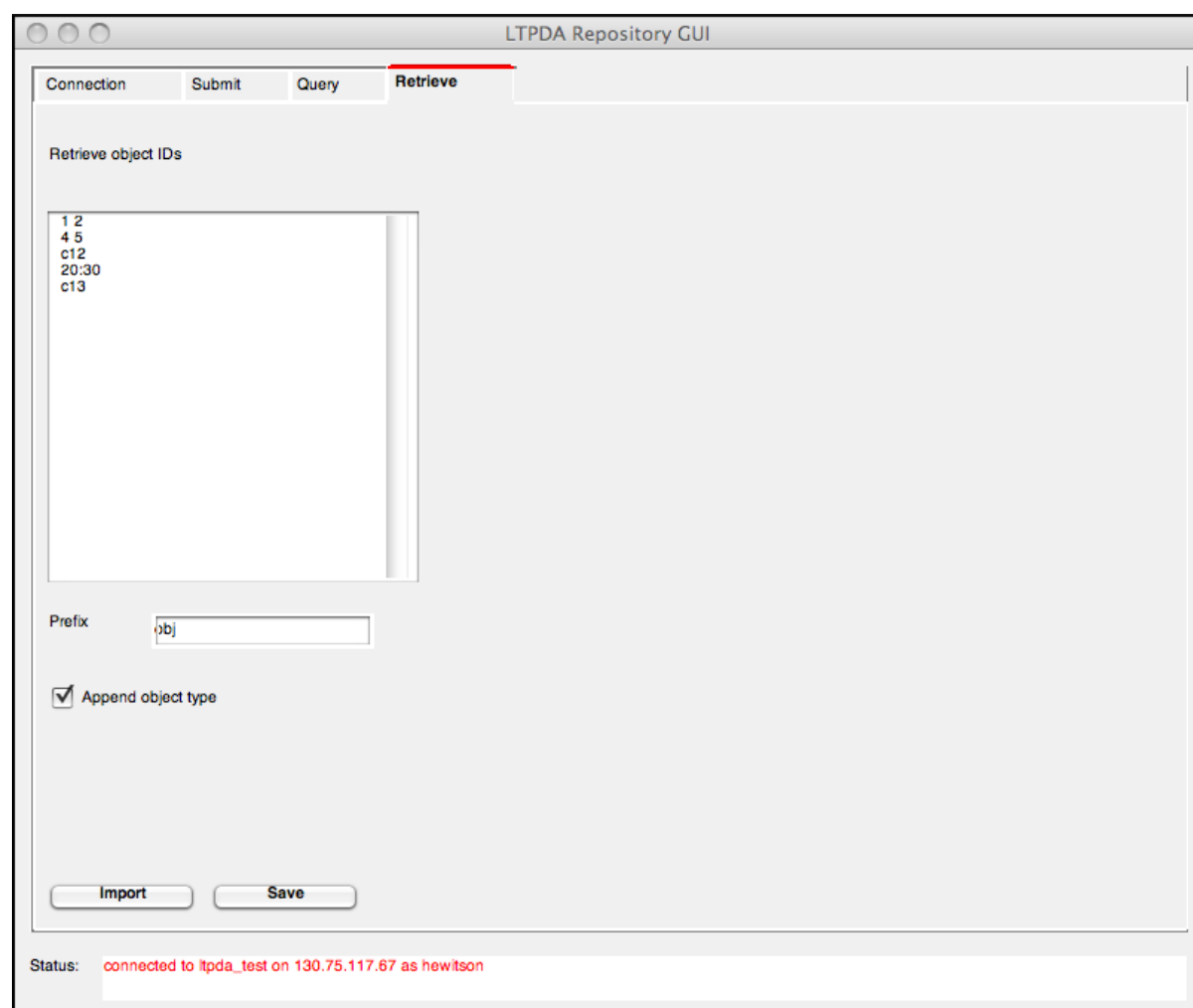
SELECT obj_id,submitted,experiment_title,experiment_desc,analysis_desc FROM objmeta WHERE submitted > "2008-02-24" AND submitted < "2008-02-25" ORDER BY obj_id DESC;

	obj_id	submitted	experiment_title	experiment_desc	analysis_desc
1	40	2008-02-24 22:16:44.0	submit timespan	this is just a test of the w...	just submitting
2	39	2008-02-24 22:16:43.0	submit time	this is just a test of the w...	just submitting
3	38	2008-02-24 22:16:42.0	submit specwin	this is just a test of the w...	just submitting
4	37	2008-02-24 22:16:41.0	submit pzmodel	this is just a test of the w...	just submitting
5	36	2008-02-24 22:16:39.0	submit plist	this is just a test of the w...	just submitting
6	35	2008-02-24 22:16:38.0	submit mlir	this is just a test of the w...	just submitting
7	34	2008-02-24 22:16:35.0	submit mfir	this is just a test of the w...	just submitting
8	33	2008-02-24 22:16:32.0	submit ao	this is just a test of the w...	just submitting
9	32	2008-02-24 21:35:59.0	Repository Test from UTN ...	Submit/retrieve test # 47...	Nothing serious, just playing ...
10	31	2008-02-24 20:41:10.0	A series of AOs	A set of AOs which are c...	No analysis yet
11	30	2008-02-24 20:41:04.0	A series of AOs	A set of AOs which are c...	No analysis yet
12	29	2008-02-24 20:40:58.0	A series of AOs	A set of AOs which are c...	No analysis yet
13	28	2008-02-24 20:40:52.0	A series of AOs	A set of AOs which are c...	No analysis yet
14	27	2008-02-24 20:40:47.0	A series of AOs	A set of AOs which are c...	No analysis yet
15	26	2008-02-24 20:40:41.0	A series of AOs	A set of AOs which are c...	No analysis yet
16	25	2008-02-24 20:40:12.0	A series of AOs	A set of AOs which are c...	No analysis yet
17	24	2008-02-24 20:40:06.0	A series of AOs	A set of AOs which are c...	No analysis yet
18	23	2008-02-24 20:40:00.0	A series of AOs	A set of AOs which are c...	No analysis yet
19	22	2008-02-24 20:39:54.0	A series of AOs	A set of AOs which are c...	No analysis yet
20	21	2008-02-24 20:39:48.0	A series of AOs	A set of AOs which are c...	No analysis yet
21	20	2008-02-24 20:39:42.0	A series of AOs	A set of AOs which are c...	No analysis yet
22	19	2008-02-24 20:35:48.0	A series of AOs	A set of AOs which are c...	No analysis yet
23	18	2008-02-24 20:35:43.0	A series of AOs	A set of AOs which are c...	No analysis yet
24	17	2008-02-24 20:35:37.0	A series of AOs	A set of AOs which are c...	No analysis yet
25	16	2008-02-24 20:35:31.0	A series of AOs	A set of AOs which are c...	No analysis yet

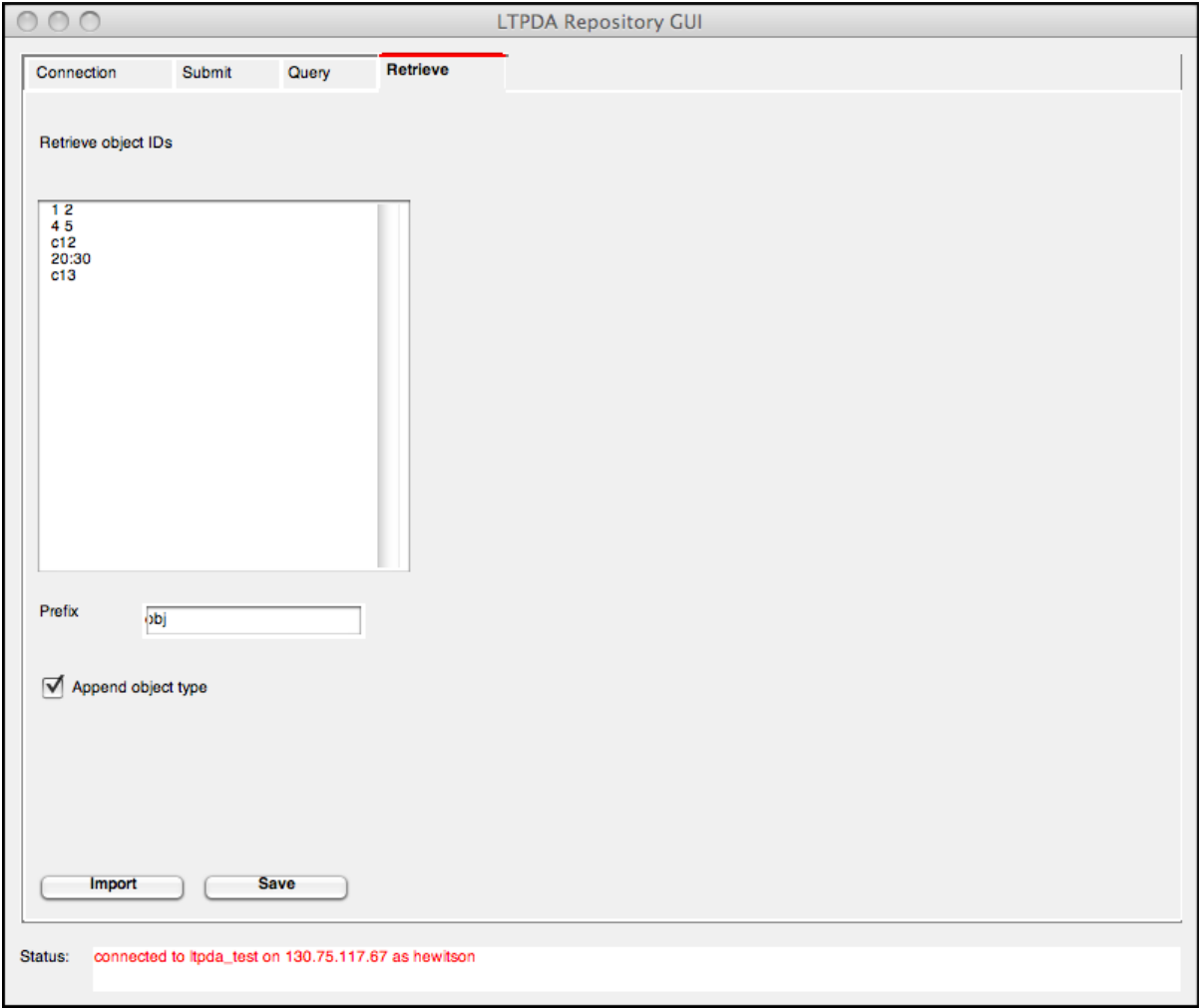
As you can see, the query string that is actually executed is presented in the text edit box above the 'Execute query' button. This query string can be edited to allow for finer control over the query. This is for users who already have a working knowledge of MySQL SQL syntax.

Retrieving objects and collections from a repository

Retrieving objects from an LTPDA repository can be done using the retrieve panel shown below:



The user must enter the IDs of the objects he/she wishes to retrieve. The IDs can be entered using standard MATLAB numerical notation. Collections can be retrieved by prefixing the collection ID with a 'c', for example, 'c12' retrieves collection 12.



Clicking on the 'Import' button retrieves all objects in the list and places them in the MATLAB workspace, as shown below:

Name ▲	Value	Min	Bytes	Class	Max
obj001_ao	<1x1 ao>		54862	ao	
obj002_ao	<1x1 ao>		40320	ao	
obj004_ao	<1x1 ao>		42666	ao	
obj005_ao	<1x1 ao>		52398	ao	
obj020_ao	<1x1 ao>		166098	ao	
obj021_ao	<1x1 ao>		166098	ao	
obj022_ao	<1x1 ao>		166098	ao	
obj023_ao	<1x1 ao>		166098	ao	
obj024_ao	<1x1 ao>		166098	ao	
obj025_ao	<1x1 ao>		166098	ao	
obj026_ao	<1x1 ao>		166098	ao	
obj027_ao	<1x1 ao>		166098	ao	
obj028_ao	<1x1 ao>		166098	ao	
obj029_ao	<1x1 ao>		166098	ao	
obj030_ao	<1x1 ao>		166098	ao	
objC012_039_time	<1x1 struct>		10402	struct	
objC013_040_timespan	<1x1 timespan>		18340	timespan	

The objects can be directly saved to disk in XML format by clicking the 'Save' button.

You can select a prefix for the objects by typing in the 'prefix' edit box.

If the 'Append object type' check-box is checked, each object name (or filename) will have the object type (class) appended.

◀ Querying the contents of a repository

Class descriptions ▶

©LTP Team

Class descriptions

AO class	Implements analysis objects in the LTPDA toolbox
SSM class	Implements statespace model in the LTPDA toolbox
MFIR class	Implements finite impulse response filter objects within LTPDA toolbox
MIIR class	Implements infinite impulse response filter objects within LTPDA toolbox
PZMODEL class	Implements pole/zero model objects within LTPDA toolbox
TIMESPAN class	Implements time span objects within LTPDA toolbox
PLIST class	Implements parameter list objects within LTPDA toolbox
SPECWIN class	Implements spectral window objects within LTPDA toolbox
TIME class	Implements time objects within LTPDA toolbox
PZ (POLE/ZERO) class	Implements pole/zero objects within LTPDA toolbox
MINFO class	Implements m-file info objects within LTPDA toolbox
HISTORY class	Implements history objects within LTPDA toolbox
PROVENANCE class	Implements provenance objects within LTPDA toolbox
PARAM class	Implements parameter objects within LTPDA toolbox
CDATA class	Implements constant data objects within LTPDA toolbox
FSDATA class	Implements frequency-series data objects within LTPDA toolbox
TSDATA class	Implements time-series data objects within LTPDA toolbox
XYDATA class	Implements x-y data objects within LTPDA toolbox

XYZDATA class	Implements x-y-z data objects within LTPDA toolbox

◀ Retrieving objects and collections from a repository	ao Class ▶

©LTP Team

ao Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties	Description	Defined in class
data	Data object associated with this AO	ao
mfile	Full text representation of the m-file that created this ao AO	
mfilename	The filename of the m-file that created this AO	ao
mdlfile	Full text representation of the mdl-file that created this AO	ao
mdlfilename	The filename of the mdl-file that created this AO	ao
procinfo	Contains extra processing information not contained in the main result of any method (plist-object).	ao
plotinfo	Plist-object which contains the	ao
description	Description of the AO	ao
hist	History object associated with this object	ltpda_uoh
name	Name of the object	ltpda_uo

created	Time object which stores the creation time of the object	ltpda_uo
creator	Creator of the object	ltpda_uo
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Arithmetic Operator	
Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
MDC1	Mock data challenge 1
Operator	Operator methods
Output	Output methods
Relational Operator	Relational operator methods
Signal Processing	Signal processing methods
Trigonometry	Trigometry methods

[▲ Back to Top](#)

Arithmetic Operator

Methods	Description	Defined in class
minus	MINUS implements minus operator for analysis objects.	ao
mpower	MPOWER implements mpower operator for analysis objects.	ao
mrdivide	MRDIVIDE implements mrdivide operator for analysis objects.	ao
mtimes	MTIMES implements mtimes operator for analysis objects.	ao
plus	PLUS implements addition operator for analysis objects.	ao
power	POWER implements power operator for analysis objects.	ao
rdivide	RDIVIDE implements rdivide operator for analysis	ao

objects.

[times](#) TIMES implements times operator for analysis objects. ao

[▲ Back to Top of Section](#)

Constructor

Methods	Description	Defined in class
ao	AO analysis object class constructor.	ao

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
attachm	ATTACHM attach an m file to the analysis object.	ao
attachmdl	ATTACHMDL attach an mdl file to the analysis object.	ao
cat	CAT concatenate AOs into a vector.	ao
copy	COPY makes a (deep) copy of the input AOs.	ao
demux	DEMUX splits the input vector of AOs into a number of output AOs.	ao
find	FIND particular samples that satisfy the input query and return a new AO.	ao
index	INDEX index into an AO array or matrix. This properly captures the history.	ao
join	JOIN multiple AOs into a single AO.	ao
len	LEN overloads the length operator for Analysis objects. Length of the data samples.	ao
md5	MD5 computes an MD5 checksum from an analysis objects.	ao
search	SEARCH selects AOs that match the given name.	ao
setDescription	SETDESCRIPTION sets the 'description' property of the ao.	ao
setFs	SETFS sets the 'fs' property of the ao.	ao

setT0	SETT0 sets the 't0' property of the ao.	ao
setX	SETX sets the 'x' property of the ao.	ao
setXY	SETXY sets the 'xy' property of the ao.	ao
setXunits	SETXUNITS sets the 'xunits' property of the ao.	ao
setY	SETY sets the 'y' property of the ao.	ao
setYunits	SETYUNITS sets the 'yunits' property of the ao.	ao
setZ	SETZ sets the 'z' property of the ao.	ao
string	STRING writes a command string that can be used to recreate the input Analysis object(s).	ao
timeshift	TIMESHIFT for AO/tsdata objects, shifts the time axis such that $x(1) = 0$.	ao
validate	VALIDATE checks that the input Analysis Object is reproducible and valid.	ao
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
setName	SETNAME Set the property 'name'.	ltpda_uoh

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
addHistory	ADDHISTORY Add a history-object to the ltpda_uo object.	ltpda_uoh

[▲ Back to Top of Section](#)

MDC1

Methods	Description	Defined in class
mdc1_ifo2acc_fd	MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.	ao
mdc1_ifo2acc_inloop	MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.	ao
mdc1_ifo2control	MDC1_IFO2CONTROL converts the input time-series to control forces.	ao

[mdc1_x2acc](#) MDC1_X2ACC converts the input time-series ao to acceleration with a time-domain filter

[▲ Back to Top of Section](#)

Operator

Methods	Description	Defined in class
abs	ABS overloads the Absolute value method for Analysis objects.	ao
complex	COMPLEX overloads the complex operator for Analysis objects.	ao
conj	CONJ overloads the conjugate operator for Analysis objects.	ao
ctranspose	CTRANSPOSE overloads the ' operator for Analysis Objects.	ao
det	DET overloads the determinant function for Analysis objects.	ao
diag	DIAG overloads the diagonal operator for Analysis Objects.	ao
eig	EIG overloads the determinant function for Analysis objects.	ao
exp	EXP overloads the exp operator for Analysis objects. Exponential.	ao
imag	IMAG overloads the imaginary operator for Analysis objects.	ao
inv	INV overloads the inverse function for Analysis Objects.	ao
ln	LN overloads the log operator for Analysis objects. Natural logarithm.	ao
log	LOG overloads the log operator for Analysis objects. Natural logarithm.	ao
log10	LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.	ao
max	MAX computes the maximum value of the data in the AO.	ao
mean	MEAN computes the mean value of the data in the AO.	ao

median	MEDIAN computes the median value of the data in the AO.	ao
min	MIN computes the minimum value of the data in the AO.	ao
mode	MODE computes the modal value of the data in the AO.	ao
norm	NORM overloads the norm operator for Analysis Objects.	ao
phase	PHASE overloads the ltpda_phase operator for Analysis objects.	ao
real	REAL overloads the real operator for Analysis objects.	ao
sign	SIGN overloads the sign operator for Analysis objects.%	ao
sort	SORT the values in the AO.	ao
sqrt	SQRT computes the square root of the data in the AO.	ao
std	STD computes the standard deviation of the data in the AO.	ao
sum	SUM computes the sum of the data in the AO.	ao
svd	SVD overloads the determinant function for Analysis objects.	ao
transpose	TRANSPOSE overloads the .' operator for Analysis Objects.	ao
uminus	UMINUS overloads the uminus operator for Analysis objects.	ao
var	VAR computes the variance of the data in the AO.	ao
zeropad	ZEROPAD zero pads the input data series.	ao

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR overloads char() function for analysis objects.	ao
display	DISPLAY implement terminal display for analysis object.	ao
export	EXPORT export an analysis object to a text file.	ao

extractm	EXTRACTM extracts an m-file from an analysis object	ao
	and saves it to disk.	
extractmdl	EXTRACTMDL extracts an mdl file from an analysis object and saves it to disk.	ao
iplot	IPLOT provides an intelligent plotting tool for LTPDA.	ao
plot	PLOT a simple plot of analysis objects.	ao
save	SAVE overloads save operator for ltpda objects.	ltpda_uo
type	TYPE converts the input objects to MATLAB functions.	ltpda_uoh

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
ge	GE overloads >= operator for analysis objects. Compare the y-axis values.	ao
gt	GT overloads > operator for analysis objects. Compare the y-axis values.	ao
le	LE overloads <= operator for analysis objects. Compare the y-axis values.	ao
lt	LT overloads < operator for analysis objects. Compare the y-axis values.	ao
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

Signal Processing

Methods	Description	Defined in class
cohere	COHERE makes coherence estimates of the time-series objects	ao
compute	COMPUTE performs the given operations on the input AOs.	ao
consolidate	CONSOLIDATE resamples all input AOs onto the same time grid.	ao

conv	CONV vector convolution.	ao
cpsd	CPSD makes cross-spectral density estimates of the time-series objects.	ao
delay	DELAY delays a time-series using various methods.	ao
detrend	DETREND detrends the input analysis object using a polynomial of degree N.	ao
dft	DFT computes the DFT of the input time-series at the requested frequencies.	ao
diff	DIFF differentiates the data in AO.	ao
dopplercorr	No description	ao
downsample	DOWNSAMPLE AOs containing time-series data.	ao
dropduplicates	DROPDUPPLICATES drops all duplicate samples in time-series AOs.	ao
dsmean	DSMEAN performs a simple downsampling by taking the mean of every N samples.	ao
fft	FFT overloads the fft method for Analysis objects.	ao
filter	FILTER overrides the filter function for analysis objects.	ao
filtfilt	FILTFILT overrides the filtfilt function for analysis objects.	ao
firwhiten	FIRWHITEN whitens the input time-series by building an FIR whitening filter.	ao
fixfs	FIXFS resamples the input time-series to have a fixed sample rate.	ao
fngen	FNGEN creates an arbitrarily long time-series based on the input PSD.	ao
gapfilling	No description	ao
hist	HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.	ao
ifft	IFFT overloads the ifft operator for Analysis objects.	ao
interp	INTERP interpolate the values in the input AO(s) at new values.	ao

interpmissing	INTERPMISSING interpolate missing samples in a time-series.	ao
lcohere	LCOHERE implement coherence estimation computed on a log frequency axis.	ao
lcpsd	LCPSD implement cross-power-spectral density estimation computed on a log frequency axis.	ao
lincom	LINCOM	ao
linedetect	LINEDETECT find spectral lines in the ao/fsdata objects.	ao
lpsd	LPSD implement the LPSD algorithm for analysis objects.	ao
ltfe	LTFE implement transfer-function estimation computed on a log frequency axis.	ao
polyfit	POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.	ao
psd	PSD makes power spectral density estimates of the time-series objects	ao
pwelch	PWELCH makes power spectral density estimates of the time-series objects	ao
resample	RESAMPLE overloads resample function for AOs.	ao
rms	RMS Calculate RMS deviation from spectrum	ao
select	SELECT select particular samples from the input AOs and return new AOs with only those samples.	ao
smoother	SMOOTHER smooths a given series of data points using the specified method.	ao
spectrogram	SPECTROGRAM computes a spectrogram of the given ao/tsdata.	ao
spikecleaning	spikecleaning detects and corrects possible spikes in analysis objects	ao
split	SPLIT split an analysis object into the specified segments.	ao
tfe	TFE makes transfer function estimates of the time-series objects.	ao
timedomainfit	TIMEDOMAINFIT uses Iscov to fit a set of time-series AOs to a target time-series AO.	ao

upsample	UPSAMPLE overloads upsample function for AOs.	ao
xcorr	XCORR makes cross-correlation estimates of the time-series	ao

[▲ Back to Top of Section](#)

Trigonometry

Methods	Description	Defined in class
acos	ACOS overloads the acos method for Analysis objects.	ao
asin	ASIN overloads the asin method for Analysis objects.	ao
atan	ATAN overloads the atan method for Analysis objects.	ao
atan2	ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse tangent.	ao
cos	COS overloads the cos operator for Analysis objects. Cosine of argument in radians.	ao
sin	SIN overloads the sin method for Analysis objects.	ao
tan	TAN overloads the tan method for Analysis objects.	ao

[▲ Back to Top of Section](#)

◀ Class descriptions

ssm Class ▶

ssm Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties	Description	Defined in class
description	Description string	ssm
amats	A matrix representing a difference/differential term in the state equation, block stored in a cell array	ssm
mmats	M matrix representing an inertial coefficient matrix in the state equation, block stored in a cell array	ssm
bmats	B matrix representing an input coefficient matrix in the state equation, block stored in a cell array	ssm
cmats	C matrix representing the state projection in the observation equation, block stored in a cell array	ssm
dmats	D matrix representing the direct feed through term in the observation equation, block stored in a cell array	ssm
isnumerical	This binary tells whether the system has numerical content only, or symbolic as well	ssm
timestep	Timestep of the difference equation. Zero means	ssm

the representation is time continuous and A defines a differential equation.

amats_handles	Cell array used in the future for nonlinear expressions in calculations	ssm
bmats_handles	Cell array used in the future for nonlinear expressions in calculations	ssm
cmats_handles	Cell array used in the future for nonlinear expressions in calculations	ssm
dmats_handles	Cell array used in the future for nonlinear expressions in calculations	ssm
inputnames	Names corresponding to each input column-block in the B/D matrices. Cell array of strings	ssm
inputvarnames	Names corresponding to each input column in each column-block of the B/D matrices. Cell array of cell array of Strings	ssm
inputsizes	Width corresponding to each input column-block in the B/D matrices. It is a double vector	ssm
Ninputs	Number of input column-blocks, it is a double	ssm
ssnames	Names corresponding to each input state-block in the A/B/C matrices. Cell array of strings	ssm
ssvarnames	Names corresponding to each variable in each input state-block in the A/B/C matrices. Cell array of cell array of Strings	ssm
sssizes	Size corresponding to each input state-block in the A/B/C matrices. It is a double vector	ssm
ssini	Initial value of the state space for simulation. Cell array of vectors	ssm
Nss	Number of state-blocks, it is a double	ssm
outputnames	Names corresponding to each output row-block in the C/D matrices. Cell array of strings	ssm
outputvarnames	Names corresponding to each output variable in each row-block in the C/D matrices. Cell array of cell array of Strings	ssm
outputsizes	Width corresponding to each output row-block in the C/D matrix. It is a double vector	ssm
Noutputs	Number of output row-blocks, it is a double	ssm

paramnames	Names of each parameter, stored as a string in a cell array	ssm
paramvalues	Nominal value of each parameter, stored as a double vector	ssm
paramsigmas	Expected variance of each parameter, stored as a double vector	ssm
Nparams	Total number of parameters	ssm
hist	History object associated with this object	ltpda_uoh
name	Name of the object	ltpda_uo
created	Time object which stores the creation time of the object	ltpda_uo
creator	Creator of the object	ltpda_uo
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
ssm	SSM statespace model class constructor.	ssm

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

setName	SETNAME Set the property 'name'.	ltpda_uoh

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
addHistory	ADDHISTORY Add a history-object to the ltpda_uo object.	ltpda_uoh
copy	COPY Make copy of ssm objects depending of the second input	ssm

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
save	SAVE overloads save operator for ltpda objects.	ltpda_uo
type	TYPE converts the input objects to MATLAB functions.	ltpda_uoh
char	CHAR convert a ssm object into a string.	ssm
display	DISPLAY display ssm object.	ssm
isstable	tells if ssm is numerically stable	ssm
string	STRING writes a command string that can be used to recreate the input statespace model object.	ssm

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

STATESPACE

Methods	Description	Defined in class

assemble	assembles embedded subsystems, with exogenous inputs	ssm
double	Convert a statespace model object to double arrays for given i/o	ssm
kalman	kalman applies Kalman filtering to a discrete ssm with given i/o	ssm
minreal	minreal gives a minimal realization of a ssm object by deleting state	ssm
modifparams	modifparams enables to options(i_options)y and substitute parameters	ssm
modiftimestep	modiftime modifies the timestep of a ssm object	ssm
modify	modify allows to exectue a string to modify a ssm object	ssm
reduce	reduce enables to do model simplification	ssm
reduce_model	reduce_model enables to do model simplification	ssm
simulate	simulate simulates a discrete ssm with given inputs	ssm
ssm2iirpz	ssm2iirpz converts a statespace model object to a miir or a pz	ssm
ssm2ss	SSM2SS converts a statespace model object to a MATLAB statespace object.	ssm

[▲ Back to Top of Section](#)

mfir Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
gd		mfir
ntaps	Number of coefficients in the filter	mfir
fs	Frequency of the filter	ltpda_filter
infile	Filename which builds the filter	ltpda_filter
a	Set of numerator coefficients	ltpda_filter
histout	Output history values to filter	ltpda_filter
hist	History object associated with this object	ltpda_uoh
name	Name of the object	ltpda_uo
created	Time object which stores the creation time of the object	ltpda_uo
creator	Creator of the object	ltpda_uo
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods
Signal Processing	Signal processing methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
mfir	MFIR FIR filter object class constructor.	mfir

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
setName	SETNAME Set the property 'name'.	ltpda_uoh
redesign	REDESIGN redesign the input filter to work for the given sample rate.	mfir

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
setHistout	SETHISTOUT Set the property 'histout'	ltpda_filter
addHistory	ADDHISTORY Add a history-object to the ltpda_uo object.	ltpda_uoh
copy	COPY Make copy of mfir objects depending of the second input	mfir

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
save	SAVE overloads save operator for ltpda objects.	ltpda_uo
type	TYPE converts the input objects to MATLAB functions.	ltpda_uoh
char	CHAR convert a mfir object into a string.	mfir
display	DISPLAY overloads display functionality for mfir objects.	mfir
string	STRING writes a command string that can be used to recreate the input filter object.	mfir

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

Signal Processing

Methods	Description	Defined in class
resp	RESP Make a frequency response of the filter.	mfir

[▲ Back to Top of Section](#)

◀ ssm Class

miir Class ▶

miir Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
b	Set of numerator coefficients	miir
histin	Input history values to filter	miir
ntaps	Number of coefficients in the filter	miir
fs	Frequency of the filter	ltpda_filter
infile	Filename which builds the filter	ltpda_filter
a	Set of numerator coefficients	ltpda_filter
histout	Output history values to filter	ltpda_filter
hist	History object associated with this object	ltpda_uoh
name	Name of the object	ltpda_uo
created	Time object which stores the creation time of the object	ltpda_uo
creator	Creator of the object	ltpda_uo

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods
Signal Processing	Signal processing methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
miir	MIIR IIR filter object class constructor.	miir

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
setName	SETNAME Set the property 'name'.	ltpda_uoh
redesign	REDESIGN redesign the input filter to work for the given sample rate.	miir

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
setHistout	SETHISTOUT Set the property 'histout'	ltpda_filter
addHistory	ADDHISTORY Add a history-object to the ltpda_uo object.	ltpda_uoh
copy	COPY Make copy of miir objects depending of the	miir

second input

[setHistin](#) SETHISTIN Set the property 'histin' miir

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
save	SAVE overloads save operator for ltpda objects.	ltpda_uo
type	TYPE converts the input objects to MATLAB functions.	ltpda_uoh
char	CHAR convert a miir object into a string.	miir
display	DISPLAY overloads display functionality for miir objects.	miir
string	STRING writes a command string that can be used to recreate the input filter object.	miir

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

Signal Processing

Methods	Description	Defined in class
resp	RESP Make a frequency response of the filter.	miir

[▲ Back to Top of Section](#)

pzmodel Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

 [Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
gain	Gain of the model	pzmodel
poles	Vector of poles (pz-objects)	pzmodel
zeros	Vector of zeros (pz-objects)	pzmodel
hist	History object associated with this object	ltpda_uoh
name	Name of the object	ltpda_uo
created	Time object which stores the creation time of the object	ltpda_uo
creator	Creator of the object	ltpda_uo
version	CVS version string of the constructor	ltpda_obj

 [Back to Top](#)

Methods

Constructor	Constructor of this class

Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Operator	Operator methods
Output	Output methods
Relational Operator	Relational operator methods
Signal Processing	Signal processing methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
pzmodel	PZMODEL constructor for pzmodel class.	pzmodel

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
setName	SETNAME Set the property 'name'.	ltpda_uoh
string	STRING writes a command string that can be used to recreate the input pzmodel object.	pzmodel

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
addHistory	ADDDHISTORY Add a history-object to the ltpda_uo object.	ltpda_uoh
copy	COPY Make copy of pzmodel objects depending of the second input	pzmodel
getlowerFreq	GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.	pzmodel
getupperFreq	GETUPPERFREQ gets the frequency of the highest pole or zero in the model.	pzmodel
pzm2ab	PZM2AB convert pzmodel to IIR filter coefficients	pzmodel

using bilinear transform.

[▲ Back to Top of Section](#)

Operator

Methods	Description	Defined in class
tomfir	TOMFIR approximates a pole/zero model with an FIR filter.	pzmodel
tomiir	TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.	pzmodel

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
save	SAVE overloads save operator for ltpda objects.	ltpda_uo
type	TYPE converts the input objects to MATLAB functions.	ltpda_uoh
char	CHAR convert a pzmodel object into a string.	pzmodel
display	DISPLAY overloads display functionality for pzmodel objects.	pzmodel

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

Signal Processing

Methods	Description	Defined in class
fngen	FNGEN creates an arbitrarily long time-series based on pzmodel the input pzmodel.	
resp	RESP returns the complex response of a pzmodel as an pzmodel Analysis Object.	

 [Back to Top of Section](#)

 miir Class	timespan Class 

©LTP Team

timespan Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
startT	TIME object of the start time	timespan
endT	TIME object of the end time	timespan
timeformat	Time format of the start/end time objects	timespan
timezone	Timezone of the start/end time objects	timespan
interval	Interval string of the start/end time	timespan
hist	History object associated with this object	ltpda_uoh
name	Name of the object	ltpda_uo
created	Time object which stores the creation time of the object	ltpda_uo
creator	Creator of the object	ltpda_uo
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
timespan	TIMESPAN timespan object class constructor.	timespan

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
setName	SETNAME Set the property 'name'.	ltpda_uoh
setEndT	SETENDT Set the property 'endT'.	timespan
setStartT	SETSTARTT Set the property 'startT'.	timespan
setTimeformat	SETTIMEFORMAT Set the property 'timeformat'.	timespan
setTimezone	SETTIMEZONE Set the property 'timezone'.	timespan

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
addHistory	ADDHISTORY Add a history-object to the ltpda_uo object.	ltpda_uoh
copy	COPY Make copy of time objects depending of the second input	timespan

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
save	SAVE overloads save operator for ltpda objects.	ltpda_uo
type	TYPE converts the input objects to MATLAB functions.	ltpda_uoh
char	CHAR convert a timespan object into a string.	timespan
display	DISPLAY overloads display functionality for timespan objects.	timespan
string	STRING writes a command string that can be used to recreate the input timespan object.	timespan

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ pzmodel Class

plist Class ▶

plist Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
params	List of param-objects (key/value pairs)	plist
name	Name of the object	ltpda_uo
created	Time object which stores the creation time of the object	ltpda_uo
creator	Creator of the object	ltpda_uo
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods

[Relational Operator](#)

Relational operator methods

[Signal Processing](#)

Signal processing methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
plist	PLIST Plist class object constructor.	plist

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
setName	SETNAME Set the property 'name'.	ltpda_uo
append	APPEND append a param-object, plist-object or a key/value pair to the parameter list.	plist
combine	COMBINE multiple parameter lists (plist objects) into a single plist.	plist
find	FIND overloads find routine for a parameter list.	plist
isparam	ISPARAM look for a given key in the parameter lists.	plist
nparams	NPARAMS returns the number of param objects in the list.	plist
pset	PSET set or add a key/value pair or a param-object into the parameter list.	plist
remove	REMOVE remove a parameter from the parameter list.	plist
string	STRING converts a plist object to a command string which will recreate the plist object.	plist

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of plist objects depending of the	plist

second input

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
save	SAVE overloads save operator for ltpda objects.	ltpda_uo
char	CHAR convert a parameter list into a string.	plist
display	DISPLAY display plist object.	plist

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

Signal Processing

Methods	Description	Defined in class
resp	RESP shadows miir/iirResp and pzmodel/resp.	plist

[▲ Back to Top of Section](#)

◀ timespan Class

specwin Class ▶

specwin Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
type	Name of the spectral window object	specwin
alpha	Alpha parameter for various window functions	specwin
psll	Peak sidelobe level	specwin
rov	Recommended overlap	specwin
nenbw	Normalised equivalent noise bandwidth	specwin
w3db	dB bandwidth in bins	specwin
flatness	Window flatness	specwin
ws	Sum of window values	specwin
ws2	Sum of squares of window values	specwin
win	Window samples (column vector)	specwin
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
specwin	% SPECWIN spectral window object class constructor.	specwin

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of specwin objects depending of the second input	specwin

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a specwin object into a string.	specwin
display	DISPLAY overloads display functionality for specwin objects.	specwin

plot	PLOT plots a specwin object.	specwin
string	STRING writes a command string that can be used to recreate the input window object.	specwin

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ plist Class	time Class ▶

time Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties	Description	Defined in class
utc_epoch_milli	Unix epoch time in milliseconds. The underlying timezone is UTC	time
timezone	Timezone of the current time	time
timeformat	Time format of the current time	time
time_str	Time string depending on the unix epoch time and time format	time
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Arithmetic Operator	
Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage

Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Arithmetic Operator

Methods	Description	Defined in class
minus	MINUS overloads – operator for time objects.	time
plus	PLUS overloads + operator for time objects.	time

[▲ Back to Top of Section](#)

Constructor

Methods	Description	Defined in class
time	TIME time object class constructor.	time

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
getTimezones	GETTIMEZONES Get all possible timezones.	time
string	STRING writes a command string that can be used to recreate the input time object.	time

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of time objects depending of the second input	time
setEpochtime	SETEPOCHTIME Set the property 'utc_epoch_milli'.	time
setTime_str	SETTIME_STR Set the property 'time_str'.	time
setTimeformat	SETTIMEFORMAT Set the property 'timeformat'.	time

setTimezone	SETTIMEZONE Set the property 'timezone'.	time

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a time object into a string.	time
display	DISPLAY overloads display functionality for time objects.	time
format	FORMAT Returns the time in specified format.	time

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ specwin Class	pz (pole/zero) Class ▶

pz (pole/zero) Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
f	Frequency of pole/zero	pz
q	Quality factor of pole/zero	pz
ri	Complex representation of pole/zero	pz
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods
Signal Processing	Signal processing methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
pz	PZ is the ltpda class that provides a common definition of poles and zeros.	pz

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of pz objects depending of the second input	pz
cp2iir	CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.	pz
cz2iir	CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.	pz
rp2iir	RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.	pz
rz2iir	RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.	pz

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a pz object into a string.	pz
display	DISPLAY display a pz object.	pz
string	STRING writes a command string that can be used to	pz

recreate the input pz object.

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

Signal Processing

Methods	Description	Defined in class
resp	RESP returns the complex response of the pz object.	pz

[▲ Back to Top of Section](#)

◀ time Class

minfo Class ▶

minfo Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description	Defined in class
mname	minfo
mclass	minfo
mpackage	minfo
mcategory	minfo
mversion	minfo
sets	minfo
plists	minfo
argsmin	minfo
argsmax	minfo
version	CVS version string of the constructor ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
minfo	MINFO a helper class for LTPDA methods.	minfo

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of minfo objects depending of the second input	minfo
setMversion	SETMVERSION Set the property 'mversion'.	minfo

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert an minfo object into a string.	minfo
display	DISPLAY display an minfo object.	minfo

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ pz (pole/zero) Class

history Class ▶

history Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties	Description	Defined in class
methodInfo	minfo-object which is created in the called method.	history
plistUsed	plist-object which is used in the called method.	history
methodInvars	Variable names which are used for the called method.	history
inhists	The older history-objects	history
proctime	Creation time of the history object (unix epoch time)	history
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage

Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
history	HISTORY History object class constructor.	history

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of history objects depending of the second input	history
getNodes	GETNODES converts a history object to a nodes structure suitable for plotting as a tree.	history
getNodes_plot	GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.	history

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a param object into a string.	history
display	DISPLAY implement terminal display for history object.	history
hist2dot	HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz	history
hist2m	HIST2M writes a new m-file that reproduces the analysis described in the history object.	history

plot	PLOT plots a history object as a tree diagram.	history
string	STRING writes a command string that can be used to recreate the input history object.	history

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ minfo Class	provenance Class ▶

provenance Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties	Description	Defined in class
creator	Current user of the LTPDA toolbox	provenance
ip	IP address of the creator	provenance
hostname	Hostname of the creator	provenance
os	Used system of the creator	provenance
matlab_version	MATLAB version	provenance
sigproc_version	Signal Processing Toolbox version	provenance
symbolic_math_version	Symbolic Math Toolbox version	provenance
ltpda_version	LTPDA Toolbox version	provenance
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
provenance	PROVENANCE constructors for provenance class.	provenance

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of provenance objects depending of the second input	provenance

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a provenance object into a string.	provenance
display	DISPLAY overload terminal display for provenance objects.	provenance
string	STRING writes a command string that can be used to provenance recreate the input provenance object.	

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ history Class

param Class ▶

©LTP Team

param Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
key	Key of the key/value pair	param
val	Val of the key/value pair	param
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
param	PARAM Parameter object class constructor.	param

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj
mux	MUX concatenate params into a vector.	param

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
copy	COPY Make copy of param objects depending of the second input	param
setKey	SETKEY Set the property 'key'.	param
setKeyVal	SETKEYVAL Set the properties 'key' and 'val'	param
setVal	SETVAL Set the property 'val'.	param

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a param object into a string.	param
display	DISPLAY display a parameter	param
display2	DISPLAY display a parameter	param
string	STRING writes a command string that can be used to recreate the input param object.	param

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

 provenance Class	cdata Class 

©LTP Team

cdata Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
xunits	Units of the x-axis	cdata
yunits	Units of the y-axis	cdata
x	Data values of the x-axis	cdata
y	Data values of the y-axis	cdata
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
cdata	CDATA is the constant data class.	cdata

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
applymethod	APPLYMETHOD applies the given method to the input cdata 2D data.	
applyoperator	APPLYOPERATOR applies the given operator to the two input data objects.	cdata
char	CHAR convert a cdata-object into a string.	cdata
copy	COPY Make copy of cdata objects depending of the second input	cdata
getX	GETX Get the property 'x'.	cdata
getY	GETY Get the property 'y'.	cdata
setX	SETX Set the property 'x'.	cdata
setXY	SETXY Set the property 'xy'.	cdata
setXunits	SETXUNITS Set the property 'xunits'.	cdata
setY	SETY Set the property 'y'.	cdata
setYunits	SETYUNITS Set the property 'yunits'.	cdata

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
display	DISPLAY implement terminal display for cdata object.	cdata

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ param Class	fsdata Class ▶

fsdata Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
t0	Time-stamp of the first data sample	fsdata
navs	Number of averages	fsdata
fs	Sample rate of data	fsdata
enbw	Equivalent noise bandwidth	fsdata
xunits	Units of the x-axis	data2D
yunits	Units of the y-axis	data2D
x	Data values of the x-axis	data2D
y	Data values of the y-axis	data2D
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
fsdata	FSDATA frequency-series object class constructor.	fsdata

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
applymethod	APPLYMETHOD applys the given method to the input data2D 2D data.	
applyoperator	APPLYOPERATOR applys the given operator to the two input data objects.	data2D
getX	GETX Get the property 'x'.	data2D
getY	GETY Get the property 'y'.	data2D
setX	SETX Set the property 'x'.	data2D
setXY	SETXY Set the property 'xy'.	data2D
setXunits	SETXUNITS Set the property 'xunits'.	data2D
setY	SETY Set the property 'y'.	data2D
setYunits	SETYUNITS Set the property 'yunits'.	data2D

[copy](#) COPY Make copy of fsdata objects depending of the fsdata second input

[setEnbw](#) SETENBW Set the property 'enbw'. fsdata

[setFs](#) SETFS Set the property 'fs'. fsdata

[setNavs](#) SETNAVS Set the property 'navs'. fsdata

[setT0](#) SETT0 Set the property 't0'. fsdata

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a ltpda_data-object into a string.	data2D
display	DISPLAY implement terminal display for fsdata object.	fsdata

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ cdata Class

tsdata Class ▶

tsdata Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
t0	Time-stamp of the first data sample	tsdata
fs	Sample rate of data	tsdata
nsecs	The length of this time-series in seconds	tsdata
xunits	Units of the x-axis	data2D
yunits	Units of the y-axis	data2D
x	Data values of the x-axis	data2D
y	Data values of the y-axis	data2D
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class

Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
tsdata	TSDATA time-series object class constructor.	tsdata

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
applymethod	APPLYMETHOD applys the given method to the input data2D 2D data.	
applyoperator	APPLYOPERATOR applys the given operator to the two input data objects.	data2D
getY	GETY Get the property 'y'.	data2D
setX	SETX Set the property 'x'.	data2D
setXY	SETXY Set the property 'xy'.	data2D
setXunits	SETXUNITS Set the property 'xunits'.	data2D
setY	SETY Set the property 'y'.	data2D
setYunits	SETYUNITS Set the property 'yunits'.	data2D
collapseX	COLLAPSEX Checks whether the x vector is evenly sampled and then removes it	tsdata

copy	COPY Make copy of tsdata objects depending of the second input	tsdata
fixNsecs	FIXNSECS fixes the numer of seconds.	tsdata
getX	GETX Get the property 'x'.	tsdata
growT	GROWT grows the time (x) vector if it is empty.	tsdata
setFs	SETFS Set the property 'fs'.	tsdata
setNsecs	SETNSECS Set the property 'nsecs'.	tsdata
setT0	SETT0 Set the property 't0'.	tsdata

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a ltpda_data-object into a string.	data2D
display	DISPLAY overloads display functionality for tsdata objects.	tsdata

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ fsdata Class

xydata Class ▶

xydata Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
xunits	Units of the x-axis	data2D
yunits	Units of the y-axis	data2D
x	Data values of the x-axis	data2D
y	Data values of the y-axis	data2D
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage
Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
xydata	XYDATA X-Y data object class constructor.	xydata

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
applymethod	APPLYMETHOD applies the given method to the input data2D 2D data.	
applyoperator	APPLYOPERATOR applies the given operator to the two input data objects.	data2D
getX	GETX Get the property 'x'.	data2D
getY	GETY Get the property 'y'.	data2D
setX	SETX Set the property 'x'.	data2D
setXY	SETXY Set the property 'xy'.	data2D
setXunits	SETXUNITS Set the property 'xunits'.	data2D
setY	SETY Set the property 'y'.	data2D
setYunits	SETYUNITS Set the property 'yunits'.	data2D
copy	COPY Make copy of xydata objects depending of the xydata second input	

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a ltpda_data-object into a string.	data2D
display	DISPLAY overloads display functionality for xydata objects.	xydata

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ tsdata Class

xyzdata Class ▶

xyzdata Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

The LTPDA toolbox restrict access of the properties.
The get access is 'public' and thus it is possible to get the values with the dot-command (similar to structures).

```
For example:  
val = obj.prop(2).prop;
```

The set access is 'protected' and thus it is only possible to assign a value to a property with a set-method.

```
For example:  
obj2 = obj1.setName('my name') % This command creates a copy of obj1 (obj1 ~= obj2)  
obj.setName('my name');        % This command applies to obj
```

Properties Description		Defined in class
zunits	Units of the z-axis	data3D
z	Data values of the z-axis	data3D
xunits	Units of the x-axis	data2D
yunits	Units of the y-axis	data2D
x	Data values of the x-axis	data2D
y	Data values of the y-axis	data2D
version	CVS version string of the constructor	ltpda_obj

[▲ Back to Top](#)

Methods

Constructor	Constructor of this class
Helper	Helper methods only for internal usage

Internal	Internal methods only for internal usage
Output	Output methods
Relational Operator	Relational operator methods

[▲ Back to Top](#)

Constructor

Methods	Description	Defined in class
xyzdata	XZYDATA X-Y-Z data object class constructor.	xyzdata

[▲ Back to Top of Section](#)

Helper

Methods	Description	Defined in class
isprop	ISPROP tests if the given field is one of the object properties.	ltpda_obj

[▲ Back to Top of Section](#)

Internal

Methods	Description	Defined in class
getX	GETX Get the property 'x'.	data2D
getY	GETY Get the property 'y'.	data2D
setX	SETX Set the property 'x'.	data2D
setXY	SETXY Set the property 'xy'.	data2D
setXunits	SETXUNITS Set the property 'xunits'.	data2D
setY	SETY Set the property 'y'.	data2D
setYunits	SETYUNITS Set the property 'yunits'.	data2D
getZ	GETZ Get the property 'z'.	data3D
setZ	SETZ Set the property 'z'.	data3D
setZunits	SETZUNITS Set the property 'zunits'.	data3D
copy	COPY Make copy of xyzdata objects depending of the second input	xyzdata

[▲ Back to Top of Section](#)

Output

Methods	Description	Defined in class
char	CHAR convert a ltpda_data-object into a string.	xyzdata
display	DISPLAY overloads display functionality for xyzdata objects.	xyzdata

[▲ Back to Top of Section](#)

Relational Operator

Methods	Description	Defined in class
eq	EQ overloads the == operator for ltpda objects.	ltpda_obj
ne	NE overloads the ~= operator for ltpda objects.	ltpda_obj

[▲ Back to Top of Section](#)

◀ xyzdata Class	Constructor Examples ▶

Constructor Examples

Constructor examples

- [Constructor examples of the AO class](#)
- [Constructor examples of the MFIR class](#)
- [Constructor examples of the MIIR class](#)
- [Constructor examples of the PLIST class](#)
- [Constructor examples of the PZMODEL class](#)
- [Constructor examples of the SPECWIN class](#)
- [Constructor examples of the TIMESPAN class](#)

Constructor examples of the AO class

[Construct empty AO](#)

[Construct an AO by loading the AO from a file](#)

[Construct an AO from a data file](#)

[Construct an AO from a parameter list object \(PLIST\)](#)

[Construct from a set of values](#)

Construct empty AO

The following example creates an empty analysis object

```
>> a = ao
M: running ao/ao
M:   empty constructor
M: running ao/display
----- ao: a -----

name:   none
creator: created by hewitson@bobmac.aei.uni-hannover.de[130.75.117.65] on MACI/7.6
(R2008a)/1.9.1 beta (R2008a)
description:
data:   None
hist:   ao / ao / $Id: constructor_examples_ao_content.html,v 1.3 2008/08/22 15:33:51
hewitson Exp $
mfilename:
mdlfilename:
-----
```

Construct an AO by loading the AO from a file

The following example creates a new analysis object by loading the analysis object from disk.

```
a = ao('a1.mat')
a = ao('a1.xml')
```

Construct an AO from a data file

The following example creates a new analysis object by loading the data in 'file.txt'. The ascii file is assumed to be an equally sampled two-column file of time and amplitude.

```
a = ao('file.txt') or
a = ao('file.dat')
```

The following example creates a new analysis object by loading the data in 'file'. The parameter list determines how the analysis object is created. The valid key/value pairs of the parameter list are:

'type'	'tsdata','fsdata','xydata'
	[default: 'tsdata']

'use_fs'	If this value is set, the x-axes is computed by the fs value. [default: empty]
'columns'	[1 2 1 4] Each pair represents the x- and y-axes (each column pair creates an analysis object). If the value 'use_fs' is set, then each column is converted to the y vector of a time-series AO. [default: [1 2]]
'comment_char'	The comment character in the file [default: '%']
'description'	To set the description in the analysis object
'...'	Every property of the data object e.g. 'name'

```
% Each pair in col represents the x- and y-axes.
% 'use_fs' is not used !!!

pl = plist('filename', 'data.dat', ...
'description', 'my ao description', ...
'type', 'xydata', ...
'xunits', 's', ...
'yunits', {'Volt', 'Hz'}, ...
'columns', [1 2 1 3], ...
'comment_char', '//');

out = ao('data.dat', pl);
out = ao(pl);
```

Another example where the time vector is specified by the sample rate (use_fs) and each column of data is converted in to a single AO.

```
% 'use_fs' is used. As such, each column in col creates
its own AO with the specified sample rate.

pl = plist('filename', 'data.dat', ...
'type', 'tsdata', ...
'use_fs', 100, ...
't0', {'14:00:00', '14:00:20', '14:00:30'}, ...
'columns', [1 2 3]);
out = ao('data.dat', pl);
out = ao(pl);
```

Construct an AO from a parameter list (plist)

Constructs an analysis object from the description given in the parameter list.

[Use the key word 'fcn'](#)

[Use the key word 'tsfcn'](#)

[Use the key word 'fsfcn'](#)

[Use the key word 'win'](#)

[Use the key word 'waveform'](#)

[Use the key word 'polyval'](#)

Use the key word 'fcn'

The following example creates an AO from the description of any valid MATLAB function. The data object is of type `cdata` (1D data).

```
pl = plist('fcn', 'randn(100,1)');
a1 = ao(pl);
```

You can pass additional parameters to the fcn as extra parameters in the parameter list:

```
pl = plist('fcn', 'a*b', 'a', 2, 'b', 1:20);
a1 = ao(pl);
```

Use the key word 'tsfcn'

Construct an AO from a function of time, `t`. The data object is of type `tsdata` (time-series data). The relevant parameters are:

'fs'	sampling frequency [default: 10 Hz]
'nsecs'	length in seconds [default: 10 s]
't0'	Start time which is associated with the time-series [default: '1970-01-01 00:00:00.000']

Example,

```
pl = plist('fs', 10, 'nsecs', 10,
'tsfcn', 'sin(2*pi*1.4*t) + 0.1*randn(size(t))', ...
't0', '1980-12-01 12:43:12');
a1 = ao(pl)
```

Use the key word 'fsfcn'

Construct an AO from a function of frequency, `f`. The data object is of type `fsdata` (frequency-series). You can also specify optional parameters:

'f1'	the initial frequency [default: 1e-9]
'f2'	the final frequency [default: 5]
'nf'	the number of frequency samples [default: 1000]
'scale'	'log' or 'lin' frequency spacing [default: 'log']

or provide a frequency vector:

'f' a vector of frequencies on which
to evaluate the function

```
pl1 = plist('fsfcn', '1./f.^2', 'scale', 'lin', 'nf', 100);
pl2 = plist('fsfcn', '1./f.^2', 'f', logspace(0,3, 1000));
a1 = ao(pl1)
a2 = ao(pl2)
```

Use the key word 'win'

Construct an AO from a spectral window object.

[List of available window functions](#)

```
pl1 = plist('win', specwin('Hanning', 100))
pl2 = plist('win', specwin('Kaiser', 10, 150));
a1 = ao(pl1)
a2 = ao(pl2)
```

Use the key word 'waveform'

Construct an AO from a waveform with the following waveform types

'sine wave' 'A', 'f', 'phi'

'noise' 'type' (can be 'Normal' or
'Uniform')

'chirp' 'f0', 'f1', 't1'

'Gaussian pulse' 'f0', 'bw'

'Square wave' 'f', 'duty'

'Sawtooth' 'f', 'width'

You can also specify additional parameters:

'fs' sampling frequency [default: 10
Hz]

'nsecs' length in seconds [default: 10 s]

You can also specify the initial time (t0) associated with the time-series by passing a parameter 't0' with a value that is a time object.

```
% Construct a sine wave
pl = plist('nsecs', 10, 'fs', 1000);
pl_w = pl.append('waveform', 'sine wave', 'phi', 30, 'f', 1.23);
out_sin = ao(pl_w)
```

```
% Construct random noise
pl_w = pl.append('waveform', 'noise', 'type', 'Normal');
out_noise1 = ao(pl_w)
% Construct uniform random noise
pl_w = append(pl, 'waveform', 'noise', 'type', 'Uniform');
out_noise2 = ao(pl_w)
% Construct a chirp waveform
pl_w = append(pl, 'waveform', 'chirp', 'f0', 1, 'f1', 50, 't1', 100);
out_chirp = ao(pl_w)
% Construct a Gaussian pulse waveform
pl_w = append(pl, 'waveform', 'Gaussian pulse', 'f0', 10, 'bw', 100);
out_gaus = ao(pl_w)
% Construct a Square wave
pl_w = append(pl, 'waveform', 'Square wave', 'f', 1, 'duty', 50);
out_square = ao(pl_w)
% Construct a Sawtooth wave
pl_w = append(pl, 'waveform', 'Sawtooth', 'width', .5, 'f', 1);
out_saw = ao(pl_w)
```

Use the key word 'polyval'

Construct an AO from a set of polynomial coefficients. The relevant parameters are:

'polyval' A set of polynomial coefficients.
[default: [-0.0001 0.02 -1 -1]]

Additional parameters:

'nsecs' Number of seconds [default: 10]

'fs' Sample rate [default: 10 s]

or

't' vector of time vertices

```
pl = plist('polyval', [1 2 3], 'Nsecs', 10, 'fs', 10);
a1 = ao(pl)
```

Construct from a set of values

The following example creates an AO from a set of values.

```
vals = [1 2 3; 4 5 6; 7 8 9];
pl = plist('vals', vals);
a1 = ao(vals)
a2 = ao(pl)
```

Constructor examples of the MFIR class

[Construct empty MFIR object](#)

[Construct a MFIR object by loading the object from a file](#)

[Construct a MFIR object from an Analysis Object](#)

[Construct a MFIR object from a parameter list \(PLIST\)](#)

[Construct a MFIR object from an existing model](#)

[Construct a MFIR object from a difference equation](#)

Construct empty MFIR object

The following example creates an empty mfir object

```
>> m = mfir
----- mfir/1 -----
    gd: []
    ntaps: 0
    fs: []
    infile:
    a: []
    histout: []
    hist: mfir [1x1 history]
    name: none
    created: 2008-08-22 14:08:33.711 [1x1 time]
    creator: created by hewitson@bobmac.aei.uni-hannover.de[130.75.117.65] on MACI/7.6
    (R2008a)/1.9.1 beta (R2008a) [1x1 provenance]
    version: $Id: constructor_examples_mfir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
    Exp $ --> $Id: constructor_examples_mfir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
    Exp $ --> $Id: constructor_examples_mfir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
    Exp $ --> $Id: constructor_examples_mfir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
    Exp $ --> $Id: constructor_examples_mfir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
    Exp $
    -----
>>
```

Construct a MFIR object by loading the object from a file

The following example creates a new mfir object by loading the mfir object from disk.

```
mf = mfir('mf.mat')
mf = mfir('mf.xml')
```

Construct a MFIR object from an Analysis Object

An FIR filter can be generated based on the magnitude of the input Analysis Object or fsdata object. In the following example an fsdata object is first generated and then passed to the mfir constructor to obtain the equivalent FIR filter.

```
a1 = ao(plist('fsfcn', '1./(50+f)', 'f', linspace(0, 500, 1000)));
a1.setFs(1000);
mf = mfir(a1);
iplot(a1, resp(mf));
```

Construct a MFIR object from a parameter list (plist)

Use the key word 'type'

Construct an MFIR of a standard type.

'type' one of the types: 'highpass',
 'lowpass', 'bandpass', 'bandreject'
 [default 'lowpass']

You can also specify optional parameters:

'gain' The gain of the filter [default: 1]

'fc' The roll-off frequency [default:
 0.1 Hz]

'fs' The sampling frequency to design
 for [default: 1 Hz]

'order' The filter order [default: 64]

'win' Specify window function used in
 filter design [default: 'Hamming']

The following example creates an order 64 highpass filter with high frequency gain 2. Filter is designed for 1 Hz sampled data and has a cut-off frequency of 0.2 Hz.

```
pl = plist('type', 'highpass', ...
          'order', 64, ...
          'gain', 2.0, ...
          'fs', 1, ...
          'fc', 0.2);
f = mfir(pl)
```

Furthermore it is possible to specify a spectral window.

```
win = specwin('Kaiser', 11, 150);
pl = plist('type', 'lowpass', ...
          'win', win, ...
          'fs', 100, ...
          'fc', 20, ...
          'order', 10);
f = mfir(pl)
```

Use the key word 'pzmodel'

The following example creates a mfir object from a pole/zero model.

'pzmodel' A pzmodel object to construct the
 filter from [default: empty
 pzmodel]

'fs' Sample rate
 [default: 8 * frequency of the
 highest pole or zero in the model]


```
pzm = pzmodel(1, [pz(1) pz(200)], pz(50));
pl = plist('pzmodel', pzm, 'fs', 1000);
mf = mfir(pl)
```

Use the key word 'ao'

Construct an MFIR based on the magnitude of the input AO/fsdata object a

'method'	the design method: 'frequency-sampling' – uses fir2() 'least-squares' – uses firls() 'Parks-McClellan' – uses firpm() [default: 'frequency-sampling']
'win'	Window function for frequency-sampling method [default: 'Hanning']
'N'	Filter order [default: 512]

The following example creates a mfir object from an analysis object.

```
a1 = ao(plist('fsfcn', '1./(50+f)', 'f', linspace(0, 500, 1000)));
a1.setFs(1000);
pl = plist('ao', a1);
mf = mfir(pl)
```

Construct a MFIR object from an existing filter

The mfir constructor also accepts as an input existing filters stored in different formats:

LISO files

```
f = mfir('foo_fir.fil')
```

XML files

```
f = mfir('foo_fir.xml')
```

MAT files

```
f = mfir('foo_fir.mat')
```

From an LTPDA repository

```
f = mfir(plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []))
```

Construct a MFIR object from a difference equation

The filter can be defined in terms of two vectors specifying the coefficients of the filter and the

sampling frequency. The following example creates a FIR filter with sampling frequency 1 Hz and the following recursive equation:

$$y[n] = -0.8x[n] + 10x[n-1]$$

```
b = [-0.8 10];  
fs = 1;  
f = mfir(b,fs)
```

◀ Constructor examples of the AO class

Constructor examples of the MIIR class ▶

©LTP Team

Constructor examples of the MIIR class

[Construct empty MIIR object](#)

[Construct a MIIR object by loading the object from a file](#)

[Construct a MIIR object from a parameter list \(PLIST\)](#)

[Construct a MIIR object from an existing model](#)

[Construct a MIIR object from a difference equation](#)

Construct empty MIIR object

The following example creates an empty miir object

```
>> f = miir
M: running miir/miir
----- miir/1 -----
      b: []
  histin: []
    ntaps: 0
      fs: []
   infile:
      a: []
  histout: []
    hist: miir [1x1 history]
    name: none
  created: 2008-08-22 14:46:49.941 [1x1 time]
 creator: created by hewitson@bobmac.aei.uni-hannover.de[130.75.117.65] on MACI/7.6
(R2008a)/1.9.1 beta (R2008a) [1x1 provenance]
version: $Id: constructor_examples_miir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
Exp $ --> $Id: constructor_examples_miir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
Exp $ --> $Id: constructor_examples_miir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
Exp $ --> $Id: constructor_examples_miir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
Exp $ --> $Id: constructor_examples_miir_content.html,v 1.3 2008/08/22 15:33:51 hewitson
Exp $
-----
```

Construct a MIIR object by loading the object from a file

The following example creates a new miir object by loading the miir object from disk.

```
f = miir('f.mat')
f = miir('f.xml')
```

Construct a MIIR object from a parameter list (plist)

Use the key word 'type'

Construct an MIIR of a standard type.

'type' one of the types: 'highpass',
 'lowpass', 'bandpass', 'bandreject'
 [default 'lowpass']

You can also specify optional parameters:

'gain'	The gain of the filter [default: 1]
'fc'	The roll-off frequency [default: 0.1 Hz]
'fs'	The sampling frequency to design for [default: 1 Hz]
'order'	The filter order [default: 1]
'ripple'	pass/stop-band ripple for bandpass and bandreject filters [default: 0.5]

The following example creates an order 64 highpass filter with high frequency gain 2. Filter is designed for 1 Hz sampled data and has a cut-off frequency of 0.2 Hz.

```
pl = plist('type', 'highpass', ...
          'order', 64, ...
          'gain', 2.0, ...
          'fs', 1, ...
          'fc', 0.2);
f = miir(pl)
```

Furthermore it is possible to specify a spectral window.

```
win = specwin('Kaiser', 11, 150);
pl = plist('type', 'lowpass', ...
          'win', win, ...
          'fs', 100, ...
          'fc', 20, ...
          'order', 10);
f = miir(pl)
```

Use the key word 'pzm'

The following example creates a miir object from a pole/zero model.

'pzmodel'	A pzmodel object to construct the filter from [default: empty pzmodel]
'fs'	Sample rate [default: 8 * frequency of the highest pole or zero in the model]

```
pzm = pzmodel(1, [pz(1) pz(200)], pz(50));
pl = plist('pzmodel', pzm, 'fs', 1000);
f = miir(pl)
```

Construct a MIIR object from an existing model

The miir constructor also accepts as an input existing models in different formats:

LISO files

```
f = miir('foo_iir.fil')
```

XML files

```
f = miir('foo_iir.xml')
```

MAT files

```
f = miir('foo_iir.mat')
```

From repository

```
f = miir(plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []))
```

Construct a MIIR object from a difference equation

Alternatively, the filter can be defined in terms of two vectors specifying the coefficients of the filter and the sampling frequency. The following example creates a IIR filter with sampling frequency 1 Hz and the following recursive equation:

$$y[n] = 0.5x[n] - 0.01x[n-1] - 0.1y[n-1]$$

```
a = [0.5 -0.01];  
b = [1 0.1]  
fs = 1;  
  
f = miir(a,b,fs)
```

◀ Constructor examples of the MFIR class

Constructor examples of the PLIST class ▶

©LTP Team

Constructor examples of the PLIST class

Parameters can be grouped together into parameter lists (plist).

[Creating parameter lists from parameters](#)

[Creating parameter lists directly](#)

[Appending parameters to a parameter list](#)

[Finding parameters in a parameter list](#)

[Removing parameters from a parameter list](#)

[Setting parameters in a parameter list](#)

[Combining multiple parameter lists](#)

Creating parameter lists from parameters.

The following code shows how to create a parameter list from individual parameters.

```
>> p1 = param('a', 1); % create first parameter
>> p2 = param('b', specwin('Hanning', 100)); % create second parameter
>> pl = plist([p1 p2]) % create parameter list
----- plist 01 -----
n params: 2
---- param 1 ----
key: A
val: 1
-----
---- param 2 ----
key: B
val: specwin
----- Hanning -----

alpha: 0
psll: 31.5
rov: 50
nenbw: 1.5
w3db: 1.4382
flatness: -1.4236
ws: 50
ws2: 37.5
win: 100

-----

-----

-----
```

Creating parameter lists directly.

You can also create parameter lists directly using the following constructor format:

```
>> pl = plist('a', 1, 'b', 'hello')
----- plist 01 -----
n params: 2
---- param 1 ----
key: A
val: 1
-----
---- param 2 ----
key: B
val: 'hello'
```

Appending parameters to a parameter list.

Additional parameters can be appended to an existing parameter list using the `append` method:

```
>> pl = append(pl, param('c', 3)) % append a third parameter
----- plist 01 -----
n params: 3
---- param 1 ----
key: A
val: 1
---- param 2 ----
key: B
val: 'hello'
---- param 3 ----
key: C
val: 3
-----
```

Finding parameters in a parameter list.

Accessing the contents of a `plist` can be achieved in two ways:

```
>> p1 = pl.params(1); % get the first parameter
>> val = find(pl, 'b'); % get the second parameter
```

If the parameter name ('key') is known, then you can use the `find` method to directly retrieve the value of that parameter.

Removing parameters from a parameter list.

You can also remove parameters from a parameter list:

```
>> pl = remove(pl, 2) % Remove the 2nd parameter in the list
>> pl = remove(pl, 'a') % Remove the parameter with the key 'a'
```

Setting parameters in a parameter list.

You can also set parameters contained in a parameter list:

```
>> pl = plist('a', 1, 'b', 'hello')
>> pl = pset(pl, 'a', 5, 'b', 'ola'); % Change the values of the parameter with the
keys 'a' and 'b'
```

Combining multiple parameter lists.

Parameter lists can be combined:

```
>> pl = combine(pl1, pl2)
```

If `pl1` and `pl2` contain a parameter with the same key name, the output `plist` contains a parameter with that name but with the value from the first parameter list input.

◀ Constructor examples of the MIIR class

Constructor examples of the PZMODEL class ▶

©LTP Team

Constructor examples of the PZMODEL class

[General information about PZMODEL objects](#)

[Construct empty PZMODEL object](#)

[Construct a PZMODEL object by loading the object from a file](#)

[Construct a PZMODEL object from gain, poles and zeros](#)

[Construct a PZMODEL object from an existing model](#)

[Construct a PZMODEL object from a parameter list \(PLIST\) object](#)

[General information about pole/zero models](#)

Construct empty PZMODEL object

The following example creates an empty pzmodel object

```
pzm = pzmodel()
---- pzmodel 1 ----
model:      None
gain :      0
pole 001:   pole(NaN)
zero 001:   zero(NaN)
-----
```

Construct a PZMODEL object by loading the object from a file

The following example creates a new pzmodel object by loading the pzmodel object from disk.

```
p = pzmodel('pzmodel.mat')
p = pzmodel('pzmodel.xml')
```

Construct a PZMODEL object from gain, poles and zeros

The following code fragment creates a pole/zero model consisting of 2 poles and 2 zeros with a gain factor of 10:

```
gain = 10;
poles = [pole(1,2) pole(40)];
zeros = [zero(10,3) zero(100)];

pzm = pzmodel(gain, poles, zeros)
---- pzmodel 1 ----
model:      None
gain :      10
pole 001:   pole(1,2)
pole 002:   pole(40)
zero 001:   zero(10,3)
```

```
zero 002: zero(100)
-----
```

It is possible to give the model direct a name.

```
gain = 10;
poles = [pole(1,2) pole(40)];
zeros = [zero(10,3) zero(100)];

pzm = pzmodel(gain, poles, zeros, 'my model name')
---- pzmodel 1 ----
model:    my model name
gain :    10
pole 001: pole(1,2)
pole 002: pole(40)
zero 001: zero(10,3)
zero 002: zero(100)
-----
```

Construct a PZMODEL object from an existing model

The pzmodel constructor also accepts as an input existing models in a LISO file format

```
pzm = pzmodel('foo.fil')
```

Construct a PZMODEL object from a parameter list (PLIST) object

Construct a PZMODEL from its definion.

'gain'	Model gain [default: 1]
'poles'	Vector of pole objects [default: empty pole]
'zeros'	Vector of zero objects [default: empty zero]
'name'	Name of model [default: 'None']

```
poles = [pole(0.1) pole(1,100)];
zeros = [zero(10,3) zero(100)];
pl = plist('name', 'my filter', 'poles', poles, 'zeros', zeros, 'gain', 10);

pzm = pzmodel(pl)
---- pzmodel 1 ----
model:    my filter
gain :    10
pole 001: pole(0.1)
pole 002: pole(1,100)
zero 001: zero(10,3)
zero 002: zero(100)
-----
```

©LTP Team

Constructor examples of the SPECWIN class

[Construct empty SPECWIN object](#)

[Construct a SPECWIN object by loading the object from a file](#)

[Construct a SPECWIN of a particular window type an length](#)

[Construct a SPECWIN Kaiser window with the prescribed psll](#)

[Construct a SPECWIN object from a parameter list \(PLIST\) object](#)

Construct empty SPECWIN object

The following example creates an empty specwin object

```
w = specwin()
----- None -----
alpha: -1
psll: -1
rov: -1
nenbw: -1
w3db: -1
flatness: -1
ws: -1
ws2: -1
win: 1
-----
```

Construct a SPECWIN object by loading the object from a file

The following example creates a new specwin object by loading the specwin object from disk.

```
p = specwin('specwin.mat')
p = specwin('specwin.xml')
```

Construct a SPECWIN of a particular window type an length

To create a spectral window object, you call the `specwin` class constructor. The following code fragment creates a 100-point Hanning window:

```
>> w = specwin('Hanning', 100)
----- Hanning -----
alpha: 0
psll: 31.5
rov: 50
nenbw: 1.5
w3db: 1.4382
```

```
flatness: -1.4236
      ws: 50
      ws2: 37.5
      win: 100
-----
```

[List of available window functions](#)

Construct a SPECWIN Kaiser window with the prescribed psll

In the special case of creating a Kaiser window, the additional input parameter, PSLL, must be supplied. For example, the following code creates a 100-point Kaiser window with -150dB peak side-lobe level:

```
>> w = specwin('Kaiser', 100, 150)
----- Kaiser -----
      alpha: 6.18029
      psll: 150
      rov: 73.3738
      nenbw: 2.52989
      w3db: 2.38506
flatness: -0.52279
      ws: 28.2558
      ws2: 20.1819
      win: 100
-----
```

Construct a SPECWIN object from a parameter list (PLIST) object

Construct a SPECWIN from its definion.

'Name'	Spectral window name [default: 'Kaiser']
'N'	Spectral window length [default: 10]
'PSLL'	Peak sidelobe length (only for Kaiser type) [default: 150]

```
pl = plist('Name', 'Kaiser', 'N', 1000, 'PSLL', 75);
w = specwin(pl)
----- Kaiser -----
      alpha: 3.21394
      psll: 75
      rov: 63.4095
      nenbw: 1.85055
      w3db: 1.75392
flatness: -0.963765
      ws: 389.305
      ws2: 280.892
      win: 1000
```

```
-----

pl = plist('Name', 'Hanning', 'N', 1000);
w =specwin(pl)
----- Hanning -----

    alpha: 0
    psll: 31.5
    rov: 50
    nenbw: 1.5
    w3db: 1.4382
flatness: -1.4236
    ws: 500
    ws2: 375
    win: 1000

-----
```

◀ Constructor examples of the PZMODEL class Constructor examples of the TIMESPAN class ▶

Constructor examples of the TIMESPAN class

[Construct empty TIMESPAN object](#)

[Construct a TIMESPAN object by loading the object from a file](#)

[Construct a TIMESPAN object with a start and end time](#)

[Construct a TIMESPAN object from a parameter list \(PLIST\) object](#)

Construct empty TIMESPAN object

The following example creates an empty timespan object

```
ts = timespan()
----- timespan 01 -----

name      : None
start     : 2008-03-30 20:00:00.000
end       : 2008-03-30 20:00:00.000
timeformat: yyyy-mm-dd HH:MM:SS.FFF
interval  :
timezone  : UTC

created   : 2008-03-30 20:00:00.000
version   : $Id: timespan.m,v 1.23 2008/03/25 10:57:49 mauro Exp
plist     : plist class
-----
```

Construct a TIMESPAN object by loading the object from a file

The following example creates a new timespan object by loading the timespan object from disk.

```
t = timespan('timespan.mat')
t = timespan('timespan.xml')
```

Construct a TIMESPAN object with a start and end time

It is possible to specify the start-/end- time either with a time-object or with a time string.

```
t1_obj = time('14:00:00');
t2_obj = time('15:00:00');
t1_str = '14:00:00';
t2_str = '15:00:00';

ts1 = timespan(t1_obj, t2_obj) % two time-objects
ts2 = timespan(t1_obj, t2_obj) % two strings
ts3 = timespan(t1_str, t2_obj) % combination of time-object and string
ts4 = timespan(t1_str, t2_str) % combination of time-object and string
```

Construct a TIMESPAN object from a parameter list (PLIST) object

Construct an TIMESPAN by its properties definition

'start'	The starting time [default: '1970-01-01 00:30:00.000']
'end'	The ending time [default: '1980-01-01 12:00:00.010']

Additional parameters:

'timezone'	Timezone (string or java object) [default: 'UTC']
'timeformat'	Time format (string) [default: 'yyyy-mm-dd HH:MM:SS.FFF']

```
t1 = time('14:00:00');
t2 = time('15:00:00');

pl1 = plist('start', t1, ...
           'end', t2);
pl2 = plist('start', t1, ...
           'end', t2, ...
           'timeformat', 'yyyy-mm-dd HH:MM:SS', ...
           'timezone', 'CET');
```

```
ts1 = timespan(pl1)
ts2 = timespan(pl2)
```


Functions – By Category

Categories

- [Constructor](#)
- [Output](#)
- [Relational Operator](#)
- [Helper](#)
- [Internal](#)
- [Operator](#)
- [Trigonometry](#)
- [Signal Processing](#)
- [MDC01](#)
- [Arithmetic Operator](#)
- [Statespace](#)
- [STATESPACE](#)

Constructor

Function name	Directory	Description
ltpda_obj	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj	% LTPDA_OBJ is the abstract ltpda base class.
ltpda_nuo	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_nuo	% LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.
ltpda_uo	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo	% LTPDA_UO is the abstract ltpda base class for ltpda user object classes.
ltpda_uoh	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
minfo	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo	% MINFO a helper class for LTPDA methods.
unit	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit	% MINFO a helper class for implementing units in LTPDA.

ao	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% AO analysis object class constructor.
param	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% PARAM Parameter object class constructor.
plist	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% PLIST Plist class object constructor.
ltpda_data	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_data	% LTPDA_DATA is the abstract base class for ltpda data objects.
data2D	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% DATA2D is the abstract base class for 2-dimensional data objects.
cdata	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% CDATA is the constant data class.
tsdata	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% TSDATA time-series object class constructor.
fsdata	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata	% FSDATA frequency-series object class constructor.
xydata	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata	% XYDATA X-Y data object class constructor.
data3D	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D	% DATA3D is the abstract base class for 3-dimensional data objects.
xyzdata	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata	% XYZDATA X-Y-Z data object class constructor.
specwin	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin	% % SPECWIN spectral window object class constructor.
ltpda_filter	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter	% LTPDA_FILTER is the abstract base class for ltpda filter objects.
mfir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% MFIR FIR filter object class constructor.
miir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% MIIR IIR filter object

		class constructor.
pz	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% PZ is the ltpda class that provides a common definition of poles and zeros.
pzmodel	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% PZMODEL constructor for pzmodel class.
provenance	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance	% PROVENANCE constructors for provenance class.
time	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% TIME time object class constructor.
timespan	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% TIMESPAN timespan object class constructor.
history	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% HISTORY History object class constructor.
ssm	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% SSM statespace model class constructor.

[Back to top](#)

Output

Function name	Directory	Description
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo	% CHAR convert an minfo object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo	% DISPLAY display an minfo object.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit	% DISPLAY display an unit object.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CHAR overloads char() function for analysis objects.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DISPLAY implement terminal display for analysis object.

export	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% EXPORT export an analysis object to a text file.
extractm	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% EXTRACTM extracts an m-file from an analysis object and saves it to disk.
extractmdl	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% EXTRACTMDL extracts an mdl file from an analysis object and saves it to disk.
iplot	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% IPLOT provides an intelligent plotting tool for LTPDA.
plot	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% PLOT a simple plot of analysis objects.
save		
type		
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% CHAR convert a param object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% DISPLAY display a parameter
display2	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% DISPLAY display a parameter
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% STRING writes a command string that can be used to recreate the input param object.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% CHAR convert a parameter list into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% DISPLAY display plist object.
save		
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% DISPLAY implement terminal display for cdata object.
char		

display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% DISPLAY overloads display functionality for tsdata objects.
char		
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata	% DISPLAY implement terminal display for fsdata object.
char		
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata	% DISPLAY overloads display functionality for xydata objects.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata	% CHAR convert a ltpda_data-object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata	% DISPLAY overloads display functionality for xyzdata objects.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin	% CHAR convert a specwin object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin	% DISPLAY overloads display functionality for specwin objects.
plot	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin	% PLOT plots a specwin object.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin	% STRING writes a command string that can be used to recreate the input window object.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% CHAR convert a mfir object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% DISPLAY overloads display functionality for mfir objects.
save		
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% STRING writes a command string that can be used to recreate the input filter object.

type		
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% CHAR convert a miir object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% DISPLAY overloads display functionality for miir objects.
save		
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% STRING writes a command string that can be used to recreate the input filter object.
type		
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% CHAR convert a pz object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% DISPLAY display a pz object.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% STRING writes a command string that can be used to recreate the input pz object.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% CHAR convert a pzmodel object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% DISPLAY overloads display functionality for pzmodel objects.
save		
type		
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance	% CHAR convert a provenance object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance	% DISPLAY overload terminal display for provenance objects.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance	% STRING writes a command string that can be used to recreate the input provenance object.

char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% CHAR convert a time object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% DISPLAY overloads display functionality for time objects.
format	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% FORMAT Returns the time in specified format.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% CHAR convert a timespan object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% DISPLAY overloads display functionality for timespan objects.
save		
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% STRING writes a command string that can be used to recreate the input timespan object.
type		
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% CHAR convert a param object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% DISPLAY implement terminal display for history object.
hist2dot	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz
hist2m	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% HIST2M writes a new m-file that reproduces the analysis described in the history object.
plot	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% PLOT plots a history object as a tree diagram.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% STRING writes a command string that can be used to

		recreate the input history object.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% CHAR convert a ssm object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% DISPLAY display ssm object.
isstable	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% tells if ssm is numerically stable
save		
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% STRING writes a command string that can be used to recreate the input statespace model object.
type		

[Back to top](#)

Relational Operator

Function name	Directory	Description
eq		
ne		
eq		
ne		
eq		
ge	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% GE overloads >= operator for analysis objects. Compare the y-axis values.
gt	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% GT overloads > operator for analysis objects. Compare the y-axis values.
le	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LE overloads <= operator for analysis objects. Compare the y-axis values.
lt	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LT overloads < operator for analysis objects. Compare the y-axis values.

ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		

ne		
eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		

[Back to top](#)

Helper

Function name	Directory	Description
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit	% STRING converts a unit object to a command string which will recreate the unit object.
attachm	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ATTACHM attach an m file to the analysis object.
attachmdl	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ATTACHMDL attach an mdl file to the analysis object.
cat	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CAT concatenate AOs into a vector.
demux	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DEMUX splits the input vector of AOs into a number of output AOs.
find	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FIND particular

		samples that satisfy the input query and return a new AO.
index	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% INDEX index into an AO array or matrix. This properly captures the history.
isprop		%ISPROP Returns true if the property exists
join	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% JOIN multiple AOs into a single AO.
len	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LEN overloads the length operator for Analysis objects. Length of the data samples.
md5	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MD5 computes an MD5 checksum from an analysis objects.
search	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SEARCH selects AOs that match the given name.
setDescription	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETDESCRIPTION sets the 'description' property of the ao.
setFs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETFS sets the 'fs' property of the ao.
setName		% SETNAME Set the property 'name'.
setT0	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETT0 sets the 't0' property of the ao.
setX	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETX sets the 'x' property of the ao.
setXY	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETXY sets the 'xy' property of the ao.
setXunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETXUNITS sets the 'xunits' property of the ao.
setY	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETY sets the 'y' property of the ao.
setYunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETYUNITS sets the

		'yunits' property of the ao.
setZ	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETZ sets the 'z' property of the ao.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% STRING writes a command string that can be used to recreate the input Analysis object(s).
timeshift	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% TIMESHIFT for AO/tsdata objects, shifts the time axis such that x(1) = 0.
validate	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% VALIDATE checks that the input Analysis Object is reproducible and valid.
isprop		%ISPROP Returns true if the property exists
mux	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% MUX concatenate params into a vector.
append	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% APPEND append a param-object, plist-object or a key/value pair to the parameter list.
combine	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% COMBINE multiple parameter lists (plist objects) into a single plist.
find	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% FIND overloads find routine for a parameter list.
isparam	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% ISPARAM look for a given key in the parameter lists.
isprop		%ISPROP Returns true if the property exists
nparams	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% NPARAMS returns the number of param objects in the list.
pset	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% PSET set or add a key/value pair or a

		param-object into the parameter list.
remove	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% REMOVE remove a parameter from the parameter list.
setName		% SETNAME Set the property 'name'.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% STRING converts a plist object to a command string which will recreate the plist object.
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
redesign	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% REDESIGN redesign the input filter to work for the given sample rate.
setName		% SETNAME Set the property 'name'.
isprop		%ISPROP Returns true if the property exists
redesign	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% REDESIGN redesign the input filter to work for the given sample rate.
setName		% SETNAME Set the property 'name'.

isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
setName		% SETNAME Set the property 'name'.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% STRING writes a command string that can be used to recreate the input pzmodel object.
isprop		%ISPROP Returns true if the property exists
getTimezones	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% GETTIMEZONES Get all possible timezones.
isprop		%ISPROP Returns true if the property exists
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% STRING writes a command string that can be used to recreate the input time object.
isprop		%ISPROP Returns true if the property exists
setEndT	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% SETENDT Set the property 'endT'.
setName		% SETNAME Set the property 'name'.
setStartT	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% SETSTARTT Set the property 'startT'.
setTimeformat	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% SETTIMEFORMAT Set the property 'timeformat'.
setTimezone	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan	% SETTIMEZONE Set the property 'timezone'.
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists

setName		% SETNAME Set the property 'name'.

[Back to top](#)

Internal

Function name	Directory	Description
setMversion	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo	% SETMVERSION Set the property 'mversion'.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
fs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FS Get the data property 'fs'.
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.
t0	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% T0 Get the data property 't0'.
x	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% X Get the data property 'x'.
xunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% XUNITS Get the data property 'xunits'.
y	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% Y Get the data property 'y'.
yunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% YUNITS Get the data property 'yunits'.
setKey	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% SETKEY Set the property 'key'.
setKeyVal	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% SETKEYVAL Set the properties 'key' and 'val'
setVal	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param	% SETVAL Set the property 'val'.
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.

applymethod	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% APPLYMETHOD applies the given method to the input 2D data.
applyoperator	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% APPLYOPERATOR applies the given operator to the two input data objects.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% CHAR convert a cdata-object into a string.
getY	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% GETY Get the property 'y'.
setY	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% SETY Set the property 'y'.
setYunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% SETYUNITS Set the property 'yunits'.
applymethod		% APPLYMETHOD to the analysis object
applyoperator		% APPLYOPERATOR to the analysis object
collapseX	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% COLLAPSEX Checks whether the x vector is evenly sampled and then removes it
fixNsecs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% FIXNSECS fixes the numer of seconds.
getX	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% GETX Get the property 'x'.
getY		% GETY Get the property 'y'.
growT	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% GROWT grows the time (x) vector if it is empty.
setFs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% SETFS Set the property 'fs'.
setNsecs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% SETNSECS Set the property 'nsecs'.
setT0	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata	% SETT0 Set the property 't0'.

setX		% SETX sets the 'x' property of the ao.
setXY		% SETXY sets the 'xy' property of the ao.
setXunits		% SETXUNITS sets the 'xunits' property of the ao.
setY		% SETY sets the 'y' property of the ao.
setYunits		% SETYUNITS sets the 'yunits' property of the ao.
applymethod		% APPLYMETHOD to the analysis object
applyoperator		% APPLYOPERATOR to the analysis object
getX		% GETX Get the property 'x'.
getY		% GETY Get the property 'y'.
setEnbw	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata	% SETENBW Set the property 'enbw'.
setFs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata	% SETFS Set the property 'fs'.
setNavs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata	% SETNAVS Set the property 'navs'.
setT0	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata	% SETT0 Set the property 't0'.
setX		% SETX sets the 'x' property of the ao.
setXY		% SETXY sets the 'xy' property of the ao.
setXunits		% SETXUNITS sets the 'xunits' property of the ao.
setY		% SETY sets the 'y' property of the ao.

setYunits		% SETYUNITS sets the 'yunits' property of the ao.
applymethod		% APPLYMETHOD to the analysis object
applyoperator		% APPLYOPERATOR to the analysis object
getX		% GETX Get the property 'x'.
getY		% GETY Get the property 'y'.
setX		% SETX sets the 'x' property of the ao.
setXY		% SETXY sets the 'xy' property of the ao.
setXunits		% SETXUNITS sets the 'xunits' property of the ao.
setY		% SETY sets the 'y' property of the ao.
setYunits		% SETYUNITS sets the 'yunits' property of the ao.
getX		% GETX Get the property 'x'.
getY		% GETY Get the property 'y'.
getZ		% GETZ Get the property 'z'.
setX		% SETX sets the 'x' property of the ao.
setXY		% SETXY sets the 'xy' property of the ao.
setXunits		% SETXUNITS sets the 'xunits' property of the ao.
setY		% SETY sets the 'y' property of the ao.

setYunits		% SETYUNITS sets the 'yunits' property of the ao.
setZ		% SETZ sets the 'z' property of the ao.
setZunits		% SETZUNITS Set the property 'zunits'.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
setHistout		% SETHISTOUT Set the property 'histout'
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
setHistin	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% SETHISTIN Set the property 'histin'
setHistout		% SETHISTOUT Set the property 'histout'
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.
cp2iir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.
cz2iir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
rp2iir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
rz2iir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% RZ2IIR Return a,b IIR filter coefficients for a real zero designed

		using the bilinear transform.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
getlowerFreq	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
getupperFreq	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
pzm2ab	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% PZM2AB convert pzmodel to IIR filter coefficients using bilinear transform.
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.
setEpochtime	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% SETEPOCHTIME Set the property 'utc_epoch_milli'.
setTime_str	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% SETTIME_STR Set the property 'time_str'.
setTimeformat	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% SETTIMEFORMAT Set the property 'timeformat'.
setTimezone	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% SETTIMEZONE Set the property 'timezone'.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.
getNodes	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% GETNODES converts a history object to a nodes structure suitable for plotting as a tree.

getNodePlot	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.

[Back to top](#)

Operator

Function name	Directory	Description
abs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ABS overloads the Absolute value method for Analysis objects.
complex	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% COMPLEX overloads the complex operator for Analysis objects.
conj	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CONJ overloads the conjugate operator for Analysis objects.
ctranspose	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CTRANSPOSE overloads the ' operator for Analysis Objects.
det	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DET overloads the determinant function for Analysis objects.
diag	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DIAG overloads the diagonal operator for Analysis Objects.
eig	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% EIG overloads the determinant function for Analysis objects.
exp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% EXP overloads the exp operator for Analysis objects. Exponential.
imag	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% IMAG overloads the imaginary operator for

		Analysis objects.
inv	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% INV overloads the inverse function for Analysis Objects.
ln	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LN overloads the log operator for Analysis objects. Natural logarithm.
log	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LOG overloads the log operator for Analysis objects. Natural logarithm.
log10	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.
max	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MAX computes the maximum value of the data in the AO.
mean	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MEAN computes the mean value of the data in the AO.
median	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MEDIAN computes the median value of the data in the AO.
min	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MIN computes the minimum value of the data in the AO.
mode	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MODE computes the modal value of the data in the AO.
norm	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% NORM overloads the norm operator for Analysis Objects.
phase	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% PHASE overloads the ltpda_phase operator for Analysis objects.
real	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% REAL overloads the real operator for Analysis objects.
sign	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SIGN overloads the sign operator for Analysis objects.%

sort	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SORT the values in the AO.
sqrt	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SQRT computes the square root of the data in the AO.
std	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% STD computes the standard deviation of the data in the AO.
sum	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SUM computes the sum of the data in the AO.
svd	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SVD overloads the determinant function for Analysis objects.
transpose	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% TRANSPOSE overloads the .' operator for Analysis Objects.
uminus	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% UMINUS overloads the uminus operator for Analysis objects.
var	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% VAR computes the variance of the data in the AO.
zeropad	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ZEROPAD zero pads the input data series.
tomfir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% TOMFIR approximates a pole/zero model with an FIR filter.
tomiir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

[Back to top](#)

Trigonometry

Function name	Directory	Description
acos	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ACOS overloads the acos method for Analysis objects.
asin	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ASIN overloads the asin method for Analysis objects.

atan	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ATAN overloads the atan method for Analysis objects.
atan2	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse tangent.
cos	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% COS overloads the cos operator for Analysis objects. Cosine of argument in radians.
sin	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SIN overloads the sin method for Analysis objects.
tan	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% TAN overloads the tan method for Analysis objects.

[Back to top](#)

Signal Processing

Function name	Directory	Description
cohere	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% COHERE makes coherence estimates of the time-series objects
compute	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% COMPUTE performs the given operations on the input AOs.
consolidate	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CONSOLIDATE resamples all input AOs onto the same time grid.
conv	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CONV vector convolution.
cpsd	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CPSD makes cross-spectral density estimates of the time-series objects.
delay	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DELAY delays a time-series using various methods.
detrend	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DETREND detrends the input analysis object using a polynomial of degree N.

dft	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DFT computes the DFT of the input time-series at the requested frequencies.
diff	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DIFF differentiates the data in AO.
dopplercorr	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FOO description for function 'foo' in one line. Necessary for lookfor functionality.
downsample	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DOWNSAMPLE AOs containing time-series data.
dropduplicates	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DROPDUPLICATES drops all duplicate samples in time-series AOs.
dsmean	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% DSMEAN performs a simple downsampling by taking the mean of every N samples.
fft	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FFT overloads the fft method for Analysis objects.
filter	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FILTER overrides the filter function for analysis objects.
filtfilt	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FILTFILT overrides the filtfilt function for analysis objects.
firwhiten	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FIRWHITEN whitens the input time-series by building an FIR whitening filter.
fixfs	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FIXFS resamples the input time-series to have a fixed sample rate.
fngen	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% FNGEN creates an arbitrarily long time-series based on the input PSD.
gapfilling	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% GAPFILLING fills

		possible gaps in data.
hist	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
ifft	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% IFFT overloads the ifft operator for Analysis objects.
interp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% INTERP interpolate the values in the input AO(s) at new values.
interpmissing	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% INTERPMISSING interpolate missing samples in a time-series.
lcohere	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LCOHERE implement coherence estimation computed on a log frequency axis.
lcpsd	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LCPSD implement cross-power-spectral density estimation computed on a log frequency axis.
lincom	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LINCOM
linedetect	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LINEDETECT find spectral lines in the ao/fsdata objects.
lpsd	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LPSD implement the LPSD algorithm for analysis objects.
ltfe	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LTFE implement transfer-function estimation computed on a log frequency axis.
polyfit	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
psd	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% PSD makes power spectral density estimates of the time-series objects

pwelch	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% PWELCH makes power spectral density estimates of the time-series objects
resample	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% RESAMPLE overloads resample function for AOs.
rms	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% RMS Calculate RMS deviation from spectrum
select	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SELECT select particular samples from the input AOs and return new AOs with only those samples.
smoother	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SMOOTHER smooths a given series of data points using the specified method.
spectrogram	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SPECTROGRAM computes a spectrogram of the given ao/tsdata.
spikecleaning	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% spikecleaning detects and corrects possible spikes in analysis objects
split	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SPLIT split an analysis object into the specified segments.
tfe	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% TFE makes transfer function estimates of the time-series objects.
timedomainfit	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
upsample	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% UPSAMPLE overloads upsample function for AOs.
xcorr	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% XCORR makes cross-correlation

		estimates of the time-series
resp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist	% RESP shadows miir/iirResp and pzmodel/resp.
resp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% RESP Make a frequency response of the filter.
resp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% RESP Make a frequency response of the filter.
resp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz	% RESP returns the complex response of the pz object.
fngen	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% FNGEN creates an arbitrarily long time-series based on the input pzmodel.
resp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel	% RESP returns the complex response of a pzmodel as an Analysis Object.

[Back to top](#)

MDC01

Function name	Directory	Description
mdc1_ifo2acc_fd	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
mdc1_ifo2acc_inloop	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.
mdc1_ifo2control	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MDC1_IFO2CONTROL converts the input time-series to control forces.
mdc1_x2acc	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter

[Back to top](#)

Arithmetic Operator

Function name	Directory	Description
minus	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MINUS implements minus operator for analysis objects.
mpower	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MPOWER implements mpower operator for analysis objects.
mrdivide	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MRDIVIDE implements mrdivide operator for analysis objects.
mtimes	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% MTIMES implements mtimes operator for analysis objects.
plus	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% PLUS implements addition operator for analysis objects.
power	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% POWER implements power operator for analysis objects.
rdivide	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% RDIVIDE implements rdivide operator for analysis objects.
times	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% TIMES implements times operator for analysis objects.
minus	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% MINUS overloads – operator for time objects.
plus	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time	% PLUS overloads + operator for time objects.

[Back to top](#)

Statespace

Function name	Directory	Description
assemble	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% assembles embedded subsystems, with exogenous inputs
modifparams	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% modifparams enables to options(i_options)y and

		substitute parameters
modify	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% modify allows to exectue a string to modify a ssm object
reduce_model	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% REDUCE_MODEL enables to do model simplification
ssm2iirpz	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% ssm2iirpz converts a statespace model object to an miir or a pzmodel

[Back to top](#)

STATESPACE

Function name	Directory	Description
double	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% Convert a statespace model object to double arrays for given i/o
kalman	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	%kalman applies Kalman filtering to a discrete ssm with given i/o
minreal	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% minreal gives a minimal realization of a ssm object by deleting state
modiftimestep	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% modiftime modifies the timestep of a ssm object
reduce	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% reduce enables to do model simplification
simulate	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	%simulate simulates a discrete ssm with given inputs
ssm2ss	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% SSM2SS converts a statespace model object to a MATLAB statespace object.

[Back to top](#)

Functions — Alphabetical List

Matlab Index

Matlab Directories

-  [classes/@ao](#)
-  [classes/@cdata](#)
-  [classes/@data2D](#)
-  [classes/@data3D](#)
-  [classes/@fsdata](#)
-  [classes/@history](#)
-  [classes/@ltpda_data](#)
-  [classes/@ltpda_filter](#)
-  [classes/@ltpda_nuo](#)
-  [classes/@ltpda_obj](#)
-  [classes/@ltpda_uo](#)
-  [classes/@ltpda_uoh](#)
-  [classes/@mfir](#)
-  [classes/@miir](#)
-  [classes/@minfo](#)
-  [classes/@param](#)
-  [classes/@plist](#)
-  [classes/@provenance](#)
-  [classes/@pz](#)
-  [classes/@pzmodel](#)
-  [classes/@specwin](#)
-  [classes/@ssm](#)
-  [classes/@time](#)
-  [classes/@timespan](#)
-  [classes/@tsdata](#)
-  [classes/@unit](#)
-  [classes/@xydata](#)
-  [classes/@xyzdata](#)
-  [m/etc](#)
-  [m/gui/@jcontrol](#)
-  [m/gui/@jcontrol/private](#)
-  [m/gui/ao_browser](#)
-  [m/gui/constructor_helper](#)
-  [m/gui/gltpda](#)
-  [m/gui/ltpdaRepoGUI](#)
-  [m/gui/ltpdaRepoGUI/callbacks](#)
-  [m/gui/ltpdalauncher](#)
-  [m/gui/ltpdv](#)
-  [m/gui/ltpdv/callbacks](#)
- [m/gui/pzmodel_designer](#)
- [m/gui/quicklook](#)
- [m/gui/specwin_viewer](#)
- [m/helper](#)

 [m/math](#)

pole Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

[▲ Back to Class descriptions](#)

Properties

Public Properties	Public means that all clients can access the member by its name.
Private Properties	Private restricts the access to the class itself. Only methods that are part of the same class can access private members.

[▲ Back to Top](#)

Public Properties

Properties	Description
name	Name of pole (POLE) object

Private Properties

Properties	Description
f	Frequency of the pole
q	Q of the pole
ri	Complex value of the pole
plist	The parameter list object which creates the POLE object
version	CVS version string of the constructor
created	Time object which stores the creation time of the POLE object

Methods

Constructor	Constructor of this class

Helper	Helper functions only for internal usage
Internal	Internal functions only for internal usage
Output	Output functions

[▲ Back to Top](#)

Constructor

Methods	Description
pole	POLE construct a pole object.

[▲ Back to Top of Section](#)

Helper

Methods	Description
cat	CAT concatenate poles into a vector.
char	CHAR convert a pole object into a string.
get	GET get pole properties.
isfield	No description
isvalid	ISVALID tests if the given pole has all the correct fields of the correct type.
set	SET set a pole property.
string	STRING writes a command string that can be used to recreate the input pole object.

[▲ Back to Top of Section](#)

Internal

Methods	Description
cp2iir	CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.
rp2iir	RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
subsasgn	SUBSASGN define index assignment for pole properties.
subsref	SUBSREF Define field name indexing for pole objects.

[▲ Back to Top of Section](#)

Output

Methods	Description
display	DISPLAY overloads display functionality for pole objects.
save	SAVE a pole object to file.

 [Back to Top of Section](#)

 [plist Class](#)

provenance Class

zero Class

Properties	Properties of the class
Methods	All Methods of the class ordered by category.
Examples	Some constructor examples

 [Back to Class descriptions](#)

Properties

Public Properties	Public means that all clients can access the member by its name.
Private Properties	Private restricts the access to the class itself. Only methods that are part of the same class can access private members.

 [Back to Top](#)

Public Properties

Properties	Description
name	Name of zero (ZERO) object

Private Properties

Properties	Description
f	Frequency of the zero
q	Q of the zero
ri	Complex value of the zero
plist	The parameter list object which creates the ZERO object
version	CVS version string of the constructor
created	Time object which stores the creation time of the ZERO object

Methods

Constructor	Constructor of this class

Helper	Helper functions only for internal usage
Internal	Internal functions only for internal usage
Output	Output functions

[▲ Back to Top](#)

Constructor

Methods	Description
zero	ZERO construct a pole object.

[▲ Back to Top of Section](#)

Helper

Methods	Description
cat	CAT concatenate zeros into a vector.
get	GET get zero properties.
isfield	No description
isvalid	ISVALID tests if the given zero has all the correct fields of the correct type.
set	SET set a zero property.
string	STRING writes a command string that can be used to recreate the input zero object.

[▲ Back to Top of Section](#)

Internal

Methods	Description
cz2iir	CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
rz2iir	RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.
subsasgn	SUBSASGN define index assignment for zero properties.
subsref	SUBSREF Define field name indexing for zero objects.

[▲ Back to Top of Section](#)

Output

Methods	Description
char	CHAR convert a zero object into a string.
display	DISPLAY overloads display functionality for zero objects.
save	SAVE a zero object to file.

[▲ Back to Top of Section](#)

◀ xyzdata Class	Constructor Examples ▶

©LTP Team

ao

PURPOSE ^

AO analysis object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
AO analysis object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: AO analysis object class constructor.
             Create an analysis object.

Possible constructors:

a = ao()           - creates an empty analysis object
a = ao('a1.xml')    - creates a new analysis object by loading the
                    analysis object from disk.
a = ao('a1.mat')    - creates a new analysis object by loading the
                    analysis object from disk.
a = ao('a1.mat')    - creates a new analysis object by loading the
                    2-column data set stored in the .MAT file.
a = ao('file.txt')  - creates a new analysis object by loading the
a = ao('file.dat')  data in 'file.txt'. The ascii file is assumed
                    to be an equally sampled two-column file of
                    time and amplitude. By default, the amplitude
                    units are taken to be Volts ('V') and the
                    time samples are assumed to be in seconds.

a = ao('file',pl)   - creates a new analysis object by loading the
                    data in 'file'. The parameter list decide how the
                    analysis object is created. The valid key values
                    of the parameter list are:
                    'type'          'tsdata','fsdata','xydata'
                                   [default: 'tsdata']
                    'use_fs'        if this value is set, the
                                   x-axes is computed by the fs value.
                                   [default: empty array]
                    'columns'       [1 2 1 4]
                                   Each pair represented the x- and y-axes.
                                   (Each column pair creates an analysis object)
                                   Is the value 'use_fs' is used then
                                   represent each column the y-axes.
                                   (Each column creates an analysis object)
                                   [default: [1 2] ]
                    'comment_char'  The comment character in the file
                                   [default: '']
                    'description'   To set the description in the analysis object
                    '...'           every property where exist a public
                                   set-function in the AO class e.g.
                                   setName, setT0, setYunits, ...
                    If the constructor creates multiple ao's it is
                    possible to give each data class its own e.g.
                    'name'. In this case the parameter list with the
                    key 'name' must have cell of the different values
                    as the name of the different data objects. e.g.
                    pl = plist('columns', [1 2 1 3], ...
                               'name',     {'name1' 'name2'}, ...
                               'xunits',   {'sec'}, ...
                               'yunits',   {'V' 'Hz'});
                    This parameter list creates two ao's with tsdata.

                    'Robust'        - set this to 'yes' to use (slow)
                                   robust data reading. Useful for
                                   complicated file formats.
                                   [default: 'yes']

NOTE: Data files with comments at the end of the lines can only be
read if there are no lines with only comments. In this case, do not
specify a comment character. If you really want to load a file like
```

this, specify the 'Robust' option; this will be very slow for large files.

```
a = ao(data)           - creates an analysis object with a data
                        - object. Data object can be one of tsdata,
                        - fsdata, cdata, xydata, xyzdata.
a = ao(data, hist)     - creates an analysis object with a data
                        - object and a history object
a = ao(specwin)        - creates an analysis object from a specwin
                        - object
a = ao(plist)          - creates an analysis object from the description
                        - given in the parameter list
```

Parameter sets for plist constructor (in order of priority):

From XML File

Construct an AO by loading it from an XML file.

```
'filename' - construct an AO from a filename.
Example: plist('filename', 'a1.xml')
[default: empty string]
```

From MAT File

Construct an AO by loading it from a MAT file.

```
'filename' - construct an AO from a filename.
Example: plist('filename', 'a1.mat')
[default: empty string]
```

From ASCII File

Construct an AO by loading it from an ASCII text file.

```
'filename' - construct an AO from a filename.
Example: plist('filename', 'a1.txt')
[default: empty string]
```

For additional parameters, see constructor ao(file, pl) above.

From Function

Construct an AO from the description of any valid MATLAB function.

```
'fcn'      - any valid MATLAB function.
Example: plist('fcn', 'randn(100,1)')

You can pass additional parameters to the fcn as extra
parameters in the parameter list:
plist('fcn', 'a*b', 'a', 2, 'b', 1:20);

** Note: case is ignored in the function specification
such the following:
plist('fcn', 'a*A/b', 'a', 2, 'B', 1:20);
results in:
2*2*[1:20]

[default: 'randn(100,1)']
```

From Values

Construct an AO from a set of values.

```
'vals'     - a set of values.
Example: plist('vals', [1 2 3])
optional parameter: repeat 'N' times
Example: plist('vals', [1 2 3], 'N', 10)

[default: vals: [1], N: [1] ]
```

OR

To produce a tsdata AO

```
'xvals'    - a set of x values.
'yvals'    - a set of y values.
```

From Time-series Function

Construct an AO from a function of time, t.

'tsfcn' - a function of time.

You can also specify optional parameters
 'fs' - sampling frequency [default: 10 Hz]
 'nsecs' - length in seconds [default: 10 s]

You can also specify the initial time (t0) associated with the time-series by passing a parameter 't0' with a value that is a time object [default: time(0)]

Example:

```
plist('fs', 10, 'nsecs', 10, ...
      'tsfcn', 'sin(2*pi*1.4*t) + 0.1*randn(size(t))', ...
      't0', time('1980-12-01 12:43:12'));
```

From Frequency-series Function

Construct an AO from a function of frequency, f.

'fsfcn' - a function of frequency, f. [default: 'f']

You can also specify optional parameters:

'f1' - the initial frequency [default: 1e-9]
 'f2' - the final frequency [default: 5]
 'nf' - the number of frequency samples [default: 1000]
 'scale' - 'log' or 'lin' frequency spacing [default: 'log']
 or provide a frequency vector:
 'f' - a vector of frequencies on which to evaluate the function [default: []]

From Window

Construct an AO from a spectral window object.

'win' - A specwin object.

This creates a cdata type AO containing the window values.
 Example: plist('win', specwin('Hanning', 100))

[default: specwin('Hanning', 100)]

From Waveform

Construct an AO from a waveform description.

'waveform' - a waveform description (see options below).

You can also specify additional parameters:

'fs' - sampling frequency [default: 10 Hz]
 'nsecs' - length in seconds [default: 10 s]

and, for the following waveform types:

'sine wave' - 'A', 'f', 'phi' (can be vectors for sum of sine waves)
 'noise' - 'type' (can be 'Normal' or 'Uniform')
 'chirp' - 'f0', 'f1', 't1' (help chirp)
 'Gaussian pulse' - 'f0', 'bw' (help gausspuls)
 'Square wave' - 'f', 'duty' (help square)
 'Sawtooth' - 'f', 'width' (help sawtooth)

You can also specify the initial time (t0) associated with the time-series by passing a parameter 't0' with a value that is a time object.

[defaults: waveform: 'sine wave', A: 1, f: 1.23, phi: 0, fs: 10, nsecs: 10, t0: time(0)]

From Repository

Construct an AO by retrieving it from an LTPDA repository.

'Hostname' - the repository hostname. Only those objects which are AOs are returned.
 [default: 'localhost'];

Additional parameters:

'Database' - The database name [default: 'ltpda']
 'ID' - A vector of object IDs. [default: []]

```
'CID' - Retrieve all AO objects from a particular
        collection.
```

From Polynomial

Construct an AO from a set of polynomial coefficients.

```
'polyval' - a set of polynomial coefficients.
            [default: [-0.0001 0.02 -1 -1] ]
```

Additional parameters:

```
'Nsecs' and 'fs' - number of seconds, and sample rate
                  [defaults: nsecs: 10, fs: 10]
or 't' - vector of time vertices
        [default: [] ]
```

```
Example: a = ao(plist('polyval', [1 2 3], 'Nsecs', 10, 'fs', 10));
```

From Plist

```
'Plist' - construct from a plist. The value passed should be a plist
         object.
         [default: empty plist]
```

Examples:

1) Normally distributed random noise time-series

```
p = plist('waveform', 'noise', 'fs', 10, 'nsecs', 1000);
rd10 = ao(p);
```

Indexing:

```
b = a(1) % where a is an array of analysis objects
d = a.data; % get the data object
h = a.hist; % get the history object
d = a.data.x(1:20); % get a matrix of data values x;
```

The following call returns an minfo object that contains information about the AO constructor:

```
>> info = ao.getInfo
```

You can get information about class methods by calling:

```
>> info = ao.getInfo(method)
```

e.g. info = ao.getInfo('pwelch')

You can also restrict the sets of parameters contained in the minfo object by calling:

```
>> info = ao.getInfo(method, set)
```

e.g., info = ao.getInfo('ao', 'From Vals')

See also tsdata, fsdata, xydata, cdata, xyzdata

M Hewitson 30-01-07

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [applymethod](#) APPLYMETHOD to the analysis object
-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [char](#) CHAR overloads char() function for analysis objects.
-  [computeDFT](#) COMPUTEDFT Computes DFT using FFT or Goertzel
-  [computeperiodogram](#) COMPUTEPERIODOGRAM Periodogram spectral estimation.
-  [conv_noisegen](#) CONV_NOISEGEN calls the matlab function conv.m to convolute poles and zeros

- from a given pzmodel
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [findFsMax](#) findFsMax Returns the max Fs of a set of AOs
-  [findShortestVector](#) findShortestVector Returns the length of the shortest vector in samples
-  [fq2fac](#) FQ2FAC is a private function and is called by ngconv.m which can be found in the
-  [fromDataInMAT](#) FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object
-  [fromDatafile](#) FROMDATAFILE Construct an ao from filename AND parameter list
-  [fromFSfcn](#) FROMFSFCN Construct an ao from a fs-function string
-  [fromFcn](#) FROMFCN Construct an ao from a function string
-  [fromPolyval](#) FROMPOLYVAL Construct a time-series ao from polynomial coefficients
-  [fromPzmodel](#) FROMPZMODEL Construct a time-series ao from polynomial coefficients
-  [fromSpecWin](#) FROMSPECWIN Construct an ao from a Spectral window
-  [fromTSfcn](#) FROMTSFCN Construct an ao from a ts-function string
-  [fromVals](#) FROMVALS Construct an ao from a value set
-  [fromWaveform](#) FROMWAVEFORM Construct an ao from a waveform
-  [fs](#) FS Get the data property 'fs'.
-  [ltf_plan](#) LTF_PLAN computes all input values needed for the LPSD and LTFE algorithms.
-  [lxspecc](#) LXSPEC performs log-scale cross-spectral analysis of various forms.
-  [mchol](#) MCHOL
-  [md5](#) MD5 computes an MD5 checksum from an analysis objects.
-  [mdc1_input_noises](#) LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.
-  [mlpsd_m](#) MLPSD_M m-file only version of the LPSD algorithm
-  [mlpsd_mex](#) MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD
-  [mltfe](#) MLTFE compute log-frequency space TF
-  [ngconv](#) NGCONV is called by the function LTPDA_NOISEGEN
-  [nginit](#) NGINIT is called by the function LTPDA_NOISEGEN
-  [ngprop](#) NGPROP is called by the function LTPDA_NOISEGEN
-  [ngsetup](#) NGSETUP is called by the function LTPDA_NOISEGEN
-  [ngsetup_vpa](#) ALGONAME = mfilename;
-  [norm](#) NORM overloads the norm operator for Analysis Objects.
-  [setDescription](#) SETDESCRIPTION sets the 'description' property of the ao.
-  [setFs](#) SETFS sets the 'fs' property of the ao.
-  [setT0](#) SETT0 sets the 't0' property of the ao.
-  [setX](#) SETX sets the 'x' property of the ao.
-  [setXY](#) SETXY sets the 'xy' property of the ao.
-  [setXunits](#) SETXUNITS sets the 'xunits' property of the ao.
-  [setY](#) SETY sets the 'y' property of the ao.
-  [setYunits](#) SETYUNITS sets the 'yunits' property of the ao.
-  [setZ](#) SETZ sets the 'z' property of the ao.
-  [t0](#) T0 Get the data property 't0'.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version
-  [welch](#) WELCH Welch spectral estimation method.
-  [welchparse](#) WELCHPARSE Parser for the PWELCH & SPECTROGRAM functions
-  [welchscale](#) WELCHSCALE scales the output of welch to be in the required units
-  [x](#) X Get the data property 'x'.
-  [xspecc](#) XSPEC performs cross-spectral analysis of various forms.
-  [xunits](#) XUNITS Get the data property 'xunits'.
-  [y](#) Y Get the data property 'y'.
-  [yunits](#) YUNITS Get the data property 'yunits'.

This function is called by:

-  **ao** AO analysis object class constructor.
-  **compute** COMPUTE performs the given operations on the input AOs.
-  **conv** CONV vector convolution.
-  **copy** COPY makes a (deep) copy of the input AOs.
-  **fngen** FNGEN creates an arbitrarily long time-series based on the input PSD.
-  **fromDatafile** FROMDATAFILE Construct an ao from filename AND parameter list
-  **gapfilling** GAPFILLING fills possible gaps in data.
-  **join** JOIN multiple AOs into a single AO.
-  **lincom** LINCOM
-  **lxspect** LXSPEC performs log-scale cross-spectral analysis of various forms.
-  **mdc1_ifo2acc_fd** MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
-  **mdc1_input_noises** LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.
-  **spectrogram** SPECTROGRAM computes a spectrogram of the given ao/tsdata.
-  **spikecleaning** spikecleaning detects and corrects possible spikes in analysis objects
-  **split** SPLIT split an analysis object into the specified segments.
-  **timedomainfit** TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
-  **xcorr** XCORR makes cross-correlation estimates of the time-series
-  **xspect** XSPEC performs cross-spectral analysis of various forms.

SUBFUNCTIONS [^]

-  `function obj = set.data(obj, val)`
-  `function obj = set.mfile(obj, val)`
-  `function obj = set.mfilename(obj, val)`
-  `function obj = set.mdlfile(obj, val)`
-  `function obj = set.mdlfilename(obj, val)`
-  `function obj = set.procinfo(obj, val)`
-  `function obj = set.plotinfo(obj, val)`
-  `function obj = set.description(obj, val)`
-  `function a = ao(varargin)`
-  `function out = SETS()`
-  `function out = VEROUT()`
-  `function ii = getInfo(varargin)`
-  `function out = getDefaultPlist(set)`

SOURCE CODE [^]

```

0001 % AO analysis object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: AO analysis object class constructor.
0005 %             Create an analysis object.
0006 %
0007 %     Possible constructors:
0008 %
0009 %         a = ao()           - creates an empty analysis object
0010 %         a = ao('a1.xml')  - creates a new analysis object by loading the
0011 %                             analysis object from disk.
0012 %         a = ao('a1.mat')  - creates a new analysis object by loading the
0013 %                             analysis object from disk.
0014 %         a = ao('a1.mat')  - creates a new analysis object by loading the
0015 %                             2-column data set stored in the .MAT file.
0016 %         a = ao('file.txt') - creates a new analysis object by loading the
0017 %                             data in 'file.txt'. The ascii file is assumed
0018 %                             to be an equally sampled two-column file of

```

```

0019 % time and amplitude. By default, the amplitude
0020 % units are taken to be Volts ('V') and the
0021 % time samples are assumed to be in seconds.
0022 % a = ao('file',pl) - creates a new analysis object by loading the
0023 % data in 'file'. The parameter list decide how the
0024 % analysis object is created. The valid key values
0025 % of the parameter list are:
0026 % 'type' 'tsdata', 'fsdata', 'xydata'
0027 % [default: 'tsdata']
0028 % 'use_fs' if this value is set, the
0029 % x-axes is computed by the fs value.
0030 % [default: empty array]
0031 % 'columns' [1 2 1 4]
0032 % Each pair represented the x- and y-axes.
0033 % (Each column pair creates an analysis
object)
0034 % Is the value 'use_fs' is used then
0035 % represent each column the y-axes.
0036 % (Each column creates an analysis object)
0037 % [default: [1 2] ]
0038 % 'comment_char' The comment character in the file
0039 % [default: '']
0040 % 'description' To set the description in the analysis
object
0041 % '...' every property where exist a public
0042 % set-function in the AO class e.g.
0043 % setName, setT0, setYunits, ...
0044 % If the constructor creates multiple ao's it is
0045 % possible to give each data class its own e.g.
0046 % 'name'. In this case the parameter list with the
0047 % key 'name' must have cell of the different values
0048 % as the name of the different data objects. e.g.
0049 % pl = plist('columns', [1 2 1 3], ...
0050 % 'name', {'name1' 'name2'}, ...
0051 % 'xunits', {'sec' 'Hz'}, ...
0052 % 'yunits', {'V' 'Hz'});
0053 % This parameter list creates two ao's with tsdata.
0054 %
0055 % 'Robust' - set this to 'yes' to use (slow)
0056 % robust data reading. Useful for
0057 % complicated file formats.
0058 % [default: 'yes']
0059 %
0060 % NOTE: Data files with comments at the end of the lines can only be
0061 % read if there are no lines with only comments. In this case, do not
0062 % specify a comment character. If you really want to load a file like
0063 % this, specify the 'Robust' option; this will be very slow for large
0064 % files.
0065 %
0066 % a = ao(data) - creates an analysis object with a data
0067 % object. Data object can be one of tsdata,
0068 % fsdata, cdata, xydata, xyzdata.
0069 % a = ao(data, hist) - creates an analysis object with a data
0070 % object and a history object
0071 % a = ao(specwin) - creates an analysis object from a specwin
0072 % object
0073 % a = ao(plist) - creates an analysis object from the description
0074 % given in the parameter list
0075 %
0076 % Parameter sets for plist constructor (in order of priority):
0077 %
0078 % From XML File
0079 % -----
0080 %
0081 % Construct an AO by loading it from an XML file.
0082 %
0083 % 'filename' - construct an AO from a filename.
0084 % Example: plist('filename', 'a1.xml')
0085 % [default: empty string]
0086 %
0087 % From MAT File
0088 % -----
0089 %
0090 % Construct an AO by loading it from a MAT file.
0091 %
0092 % 'filename' - construct an AO from a filename.
0093 % Example: plist('filename', 'a1.mat')
0094 % [default: empty string]
0095 %
0096 % From ASCII File
0097 % -----
0098 %
0099 % Construct an AO by loading it from an ASCII text file.
0100 %
0101 % 'filename' - construct an AO from a filename.
0102 % Example: plist('filename', 'a1.txt')

```



```

0103 % [default: empty string]
0104 %
0105 % For additional parameters, see constructor ao(file, pl) above.
0106 %
0107 % From Function
0108 % -----
0109 %
0110 % Construct an AO from the description of any valid MATLAB function.
0111 %
0112 % 'fcn' - any valid MATLAB function.
0113 % Example: plist('fcn', 'randn(100,1)')
0114 %
0115 % You can pass additional parameters to the fcn as extra
0116 % parameters in the parameter list:
0117 % plist('fcn', 'a*b', 'a', 2, 'b', 1:20);
0118 %
0119 % ** Note: case is ignored in the function specification
0120 % such the following:
0121 % plist('fcn', 'a*A/b', 'a', 2, 'B', 1:20);
0122 % results in:
0123 % 2*2*[1:20]
0124 %
0125 % [default: 'randn(100,1)']
0126 %
0127 % From Values
0128 % -----
0129 %
0130 % Construct an AO from a set of values.
0131 %
0132 % 'vals' - a set of values.
0133 % Example: plist('vals', [1 2 3])
0134 % optional parameter: repeat 'N' times
0135 % Example: plist('vals', [1 2 3], 'N', 10)
0136 %
0137 % [default: vals: [1], N: [1] ]
0138 %
0139 % OR
0140 % To produce a tsdata AO
0141 %
0142 % 'xvals' - a set of x values.
0143 % 'yvals' - a set of y values.
0144 %
0145 %
0146 %
0147 % From Time-series Function
0148 % -----
0149 %
0150 % Construct an AO from a function of time, t.
0151 %
0152 % 'tsfcn' - a function of time.
0153 %
0154 % You can also specify optional parameters
0155 % 'fs' - sampling frequency [default: 10 Hz]
0156 % 'nsecs' - length in seconds [default: 10 s]
0157 %
0158 % You can also specify the initial time (t0) associated with
0159 % the time-series by passing a parameter 't0' with a value
0160 % that is a time object [default: time(0)]
0161 % Example:
0162 % plist('fs', 10, 'nsecs', 10, ...
0163 % 'tsfcn', 'sin(2*pi*1.4*t) + 0.1*randn(size(t))', ...
0164 % 't0', time('1980-12-01 12:43:12'));
0165 %
0166 % From Frequency-series Function
0167 % -----
0168 %
0169 % Construct an AO from a function of frequency, f.
0170 %
0171 % 'fsfcn' - a function of frequency, f. [default: 'f']
0172 %
0173 % You can also specify optional parameters:
0174 % 'f1' - the initial frequency [default: 1e-9]
0175 % 'f2' - the final frequency [default: 5]
0176 % 'nf' - the number of frequency samples [default: 1000]
0177 % 'scale' - 'log' or 'lin' frequency spacing [default: 'log']
0178 % or provide a frequency vector:
0179 % 'f' - a vector of frequencies on which to evaluate the
0180 % function [default: [] ]
0181 %
0182 % From Window
0183 % -----
0184 %
0185 % Construct an AO from a spectral window object.
0186 %
0187 % 'win' - A specwin object.
0188 %

```

```

0189 % This creates a cdata type AO containing the window values.
0190 % Example: plist('win', specwin('Hanning', 100))
0191 %
0192 % [default: specwin('Hanning', 100)]
0193 %
0194 % From Waveform
0195 % -----
0196 %
0197 % Construct an AO from a waveform description.
0198 %
0199 % 'waveform' - a waveform description (see options below).
0200 %
0201 % You can also specify additional parameters:
0202 % 'fs' - sampling frequency [default: 10 Hz]
0203 % 'nsecs' - length in seconds [default: 10 s]
0204 %
0205 % and, for the following waveform types:
0206 % 'sine wave' - 'A', 'f', 'phi' (can be vectors for sum
0207 % of sine waves)
0208 % 'noise' - 'type' (can be 'Normal' or 'Uniform')
0209 % 'chirp' - 'f0', 'f1', 't1' (help chirp)
0210 % 'Gaussian pulse' - 'f0', 'bw' (help gausspuls)
0211 % 'Square wave' - 'f', 'duty' (help square)
0212 % 'Sawtooth' - 'f', 'width' (help sawtooth)
0213 %
0214 % You can also specify the initial time (t0) associated with
0215 % the time-series by passing a parameter 't0' with a value
0216 % that is a time object.
0217 %
0218 % [defaults: waveform: 'sine wave', A: 1, f: 1.23, phi: 0,
0219 % fs: 10, nsecs: 10, t0: time(0) ]
0220 %
0221 %
0222 % From Repository
0223 % -----
0224 %
0225 % Construct an AO by retrieving it from an LTPDA repository.
0226 %
0227 % 'Hostname' - the repository hostname. Only those objects which
0228 % are AOs are returned.
0229 % [default: 'localhost'];
0230 %
0231 % Additional parameters:
0232 %
0233 % 'Database' - The database name [default: 'ltpda']
0234 % 'ID' - A vector of object IDs. [default: []]
0235 % 'CID' - Retrieve all AO objects from a particular
0236 % collection.
0237 %
0238 % From Polynomial
0239 % -----
0240 %
0241 % Construct an AO from a set of polynomial coefficients.
0242 %
0243 % 'polyval' - a set of polynomial coefficients.
0244 % [default: [-0.0001 0.02 -1 -1] ]
0245 %
0246 % Additional parameters:
0247 % 'Nsecs' and 'fs' - number of seconds, and sample rate
0248 % [defaults: nsecs: 10, fs: 10]
0249 % or 't' - vector of time vertices
0250 % [default: [] ]
0251 %
0252 % Example: a = ao(plist('polyval', [1 2 3], 'Nsecs', 10, 'fs', 10));
0253 %
0254 %
0255 % From Plist
0256 % -----
0257 %
0258 % 'Plist' - construct from a plist. The value passed should be a plist
0259 % object.
0260 % [default: empty plist]
0261 %
0262 %
0263 %
0264 % Examples:
0265 %
0266 % 1) Normally distributed random noise time-series
0267 %
0268 % p = plist('waveform', 'noise', 'fs', 10, 'nsecs', 1000);
0269 % rd10 = ao(p);
0270 %
0271 %
0272 % Indexing:
0273 % b = a(1) % where a is an array of analysis objects
0274 % d = a.data; % get the data object

```

```

0275 %         h = a.hist;           % get the history object
0276 %         d = a.data.x(1:20); % get a matrix of data values x;
0277 %
0278 % The following call returns an minfo object that contains information
0279 % about the AO constructor:
0280 %
0281 % >> info = ao.getInfo
0282 %
0283 % You can get information about class methods by calling:
0284 %
0285 % >> info = ao.getInfo(method)
0286 %
0287 % e.g. info = ao.getInfo('pwlch')
0288 %
0289 % You can also restrict the sets of parameters contained in the minfo
0290 % object by calling:
0291 %
0292 % >> info = ao.getInfo(method, set)
0293 %
0294 % e.g., info = ao.getInfo('ao', 'From Vals')
0295 %
0296 % See also tsdata, fsdata, xydata, cdata, xyzdata
0297 %
0298 % M Hewitson 30-01-07
0299 %
0300 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0301 %
0302 %
0303 classdef ao < ltpda_uoh
0304 %
0305 %-----
0306 %----- Public (read/write) Properties -----
0307 %-----
0308 %-----
0309 properties
0310 end % End (read/write) Properties
0311 %-----
0312 %----- Private read-only Properties -----
0313 %-----
0314 %-----
0315 properties (SetAccess = protected)
0316     data          = [];
0317     mfile          = '';
0318     mfilename      = '';
0319     mdlfile        = '';
0320     mdlfilename    = '';
0321     procinfo       = plist();
0322     plotinfo       = plist();
0323     description    = '';
0324     version        = '$Id: ao.m,v 1.158 2008/09/07 18:11:06 hewitson Exp $';
0325 end % End read only properties
0326 %
0327 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0328 % Check property setting %
0329 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0330 %
0331 methods
0332     function obj = set.data(obj, val)
0333         if ~(isa(val, 'ltpda_data') || isempty(val))
0334             error('### The value for the property ''data'' must be a ltpda_data object');
0335         end
0336         obj.data = val;
0337     end
0338     function obj = set.mfile(obj, val)
0339         if ~ischar(val)
0340             error('### The value for the property ''mfile'' must be a string');
0341         end
0342         obj.mfile = val;
0343     end
0344     function obj = set.mfilename(obj, val)
0345         if ~ischar(val)
0346             error('### The value for the property ''mfilename'' must be a string');
0347         end
0348         obj.mfilename = val;
0349     end
0350     function obj = set.mdlfile(obj, val)
0351         if ~ischar(val)
0352             error('### The value for the property ''mdlfile'' must be a string');
0353         end
0354         obj.mdlfile = val;
0355     end
0356     function obj = set.mdlfilename(obj, val)
0357         if ~ischar(val)
0358             error('### The value for the property ''mdlfilename'' must be a string');
0359         end
0360         obj.mdlfilename = val;

```



```

0361 end
0362 function obj = set.procinfo(obj, val)
0363     if ~isa(val, 'plist')
0364         error('### The value for the property 'procinfo' should be a plist.');
```

```

0365     end
0366     obj.procinfo = val;
0367 end
0368 function obj = set.plotinfo(obj, val)
0369     if ~isa(val, 'plist')
0370         error('### The value for the property 'plotinfo' should be a plist.');
```

```

0371     end
0372     obj.plotinfo = val;
0373 end
0374 function obj = set.description(obj, val)
0375     if ~ischar(val)
0376         error('### The value for the property 'description' should be a string.');
```

```

0377     end
0378     obj.description = val;
0379 end
0380 end
0381
0382
0383 %-----
0384 %----- Private Properties -----
0385 %-----
0386 properties (GetAccess = protected, SetAccess = protected)
0387
0388 end
0389
0390 methods
0391     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0392     %                                     Constructor
0393     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0394     function a = ao(varargin)
0395
0396         import utils.const.*
0397         utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0398
0399         % Check the supported version
0400         utils.helper.checkMatlabVersion;
0401
0402         %%% Call superclass
0403         a = @ltpda_uoh(varargin{:});
0404
0405         %%% Set default values
0406
0407         %%% Collect all plists and combine them.
0408         [pli, invars, args] = utils.helper.collect_objects(varargin, 'plist');
0409
0410         if ~isempty(pli)
0411             pli = pli.combine();
0412             %%% Append the plist to the input-arguments
0413             args{end+1} = pli;
0414         end
0415
0416         %%% Execute appropriate constructor
0417         switch numel(args)
0418             case 0
0419                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0420                 %                                     no input
0421                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0422                 utils.helper.msg(msg.PROC1, 'empty constructor');
0423                 a.addHistory(ao.getInfo('ao', 'None'), [], [], []);
0424
0425             case 1
0426                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0427                 %                                     one input
0428                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0429
0430                 if ischar(args{1})
0431                     %%% a1 = ao('foo.mat')
0432                     %%% a1 = ao('foo.xml')
0433                     %%% a1 = ao('foo.txt')
0434                     %%% a1 = ao('foo.dat')
```

```

0435
0436                 utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
0437                 a = fromFile(a, args{1});
0438
0439                 elseif isa(args{1}, 'ao')
0440                     %%% a1 = ao(ao)
0441
0442                     utils.helper.msg(msg.PROC1, 'copying %s', args{1}.name);
0443                     a = copy(args{1}, 1);
0444                     for kk = 1:numel(args{1})
0445                         a(kk).addHistory(ao.getInfo('ao', 'None'), [], [], a(kk).hist);
0446                     end

```

```

0447
0448 elseif isstruct(args{1})
0449     %%%%%%%%%% a1 = ao(struct) %%%%%%%%%%
0450
0451     utils.helper.msg(msg.PROC1, 'constructing from struct');
0452     %% Set properties which are declared in this class
0453     astruct = args{1};
0454
0455     if isfield(astruct, 'data')
0456         %% data -> data-object
0457         if isstruct(astruct.data)
0458             a.data = utils.helper.struct2obj(astruct.data);
0459         else
0460             a.data = astruct.data;
0461         end
0462     end
0463     a.mfile = astruct.mfile;
0464     a.mfilename = astruct.mfilename;
0465     a.mdlfile = astruct.mdlfile;
0466     a.mfilename = astruct.mfilename;
0467     a.proclinfo = utils.helper.struct2obj(astruct.proclinfo, 'plist');
0468     a.plotinfo = utils.helper.struct2obj(astruct.plotinfo, 'plist');
0469     a.description = astruct.description;
0470     a.version = astruct.version;
0471
0472 elseif isnumeric(args{1})
0473     %%%%%%%%%% a1 = ao(constant) %%%%%%%%%%
0474     %%%%%%%%%% a1 = ao([1 2; 3 4]) %%%%%%%%%%
0475
0476     utils.helper.msg(msg.PROC1, 'constructing from values');
0477     a = fromVals(a, plist('VALS', args{1}));
0478
0479 elseif isa(args{1}, 'plist')
0480     %%%%%%%%%% a1 = ao(plist-object) %%%%%%%%%%
0481
0482     utils.helper.msg(msg.PROC1, 'constructing from plist');
0483     pl = args{1};
0484     filename = find(pl, 'filename');
0485     fcn = find(pl, 'fcn');
0486     vals = find(pl, 'vals');
0487     xvals = find(pl, 'xvals');
0488     yvals = find(pl, 'yvals');
0489     tsfcn = find(pl, 'tsfcn');
0490     fsfcn = find(pl, 'fsfcn');
0491     win = find(pl, 'win');
0492     waveform = find(pl, 'waveform');
0493     polycoeffs = find(pl, 'polyval');
0494     hostname = find(pl, 'hostname');
0495     conn = find(pl, 'conn');
0496     ipl = find(pl, 'Plist');
0497     pzm = find(pl, 'pzmmodel');
0498
0499     if ~isempty(filename)
0500
0501         %-----
0502         %--- Construct from file
0503         %-----
0504         utils.helper.msg(msg.PROC2, 'constructing from file %s', filename);
0505         a = fromFile(a, args{1});
0506
0507     elseif ~isempty(fcn)
0508
0509         %-----
0510         %--- Construct from function
0511         %-----
0512         utils.helper.msg(msg.PROC2, 'constructing from function %s', fcn);
0513         a = fromFcn(a, pl);
0514
0515     elseif ~isempty(vals)
0516
0517         %-----
0518         %--- Construct from Values
0519         %-----
0520         utils.helper.msg(msg.PROC2, ['constructing from values ' mat2str(vals)]);
0521         a = a.fromVals(pl);
0522
0523     elseif ~isempty(xvals) && ~isempty(yvals)
0524
0525         %-----
0526         %--- Construct from X and Y Values
0527         %-----
0528         utils.helper.msg(msg.PROC2, 'constructing from X and Y values');
0529         a = fromVals(a, pl);
0530
0531     elseif ~isempty(tsfcn)
0532

```

```

0533 %----- Construct from Time-series function
0534 %--- Construct from Time-series function
0535 %-----
0536 utils.helper.msg(msg.PROC2, 'constructing from fcn(t) %s', tsfcn);
0537 a = fromTSfcn(a, pl);
0538
0539 elseif ~isempty(fsfcn)
0540
0541 %----- Construct from frequency-series function
0542 %--- Construct from frequency-series function
0543 %-----
0544 utils.helper.msg(msg.PROC2, 'constructing from fcn(f) %s', fsfcn);
0545 a = a.fromFSfcn(pl);
0546
0547 elseif ~isempty(win)
0548
0549 %----- Construct from Window
0550 %--- Construct from Window
0551 %-----
0552 utils.helper.msg(msg.PROC2, 'constructing from window %s', char(win));
0553 a = a.fromSpecWin(pl);
0554
0555 elseif ~isempty(waveform)
0556
0557 %----- Construct from Waveform
0558 %--- Construct from Waveform
0559 %-----
0560 utils.helper.msg(msg.PROC2, 'constructing from waveform %s', waveform);
0561 a = fromWaveform(a, pl);
0562
0563 elseif ~isempty(hostname) || ~isempty(conn)
0564
0565 %----- Construct from repository
0566 %--- Construct from repository
0567 %-----
0568 utils.helper.msg(msg.PROC2, 'constructing from repository %s', hostname);
0569 a = a.fromRepository(pl);
0570
0571 elseif ~isempty(polycoeffs)
0572
0573 %----- Construct from polynomial
0574 %--- Construct from polynomial
0575 %-----
0576 utils.helper.msg(msg.PROC2, ['constructing from polynomial '
mat2str(polycoeffs)]);
0577 a = a.fromPolyval(pl);
0578
0579 elseif ~isempty(ipl)
0580
0581 %----- Construct from plist
0582 %--- Construct from plist
0583 %-----
0584 % if the plist is empty, we return an empty ao
0585 if nparams(ipl) == 0
0586 else
0587 % do plist constructor
0588 a = ao(ipl);
0589 end
0590
0591 elseif ~isempty(pzm)
0592
0593 %----- Construct from pzmodel
0594 %--- Construct from pzmodel
0595 %-----
0596 utils.helper.msg(msg.PROC2, 'constructing from pzmodel %s', char(pzm));
0597 a = a.fromPzmodel(pl);
0598
0599 else
0600
0601 %----- ERROR
0602 %--- ERROR
0603 %-----
0604 if nparams(pl) == 0
0605 %%% is the plist is empty then return an empty AO
0606 a = ao();
0607 else
0608 error('### Unknown AO constructor method.');
```

```

0614 char(args{1}));
0615 a = a.fromSpecWin(args{1});
0616
```

```

0617 elseif isa(args{1}, 'ltpda_data')
0618     %%%%%%%%%%% a1 = ao(ltpda_data-object) %%%%%%%%%%%
0619     %%%%%%%%%%% a1 = ao(cdata) %%%%%%%%%%%
0620     %%%%%%%%%%% a1 = ao(fsddata) %%%%%%%%%%%
0621     %%%%%%%%%%% a1 = ao(tsddata) %%%%%%%%%%%
0622     %%%%%%%%%%% a1 = ao(xydata) %%%%%%%%%%%
0623     %%%%%%%%%%% a1 = ao(xyzdata) %%%%%%%%%%%
0624
0625     utils.helper.msg(msg.PROC1, 'constructing from data object %s',
class(args{1}));
0626     a = ao;
0627     a.data = args{1};
0628     a.addHistory(ao.getInfo('ao', 'None'), [], [], []);
0629
0630 else
0631     error('### Unknown single input constructor');
0632 end
0633
0634 case 2
0635     %%%%%%%%%%% two inputs %%%%%%%%%%%
0636     %%%%%%%%%%% two inputs %%%%%%%%%%%
0637     %%%%%%%%%%% two inputs %%%%%%%%%%%
0638
0639     if isa(varargin{1}, 'database')
0640         %%%%%%%%%%% ao(database-object, [IDs]) %%%%%%%%%%%
0641
0642         utils.helper.msg(msg.PROC1, 'constructing from database object');
0643         pl = plist('conn', varargin{1}, 'id', varargin{2});
0644         a = a.fromRepository(pl);
0645
0646     elseif isnumeric(args{1}) && isnumeric(args{2})
0647         %%%%%%%%%%% ao(x-vector, y-vector) %%%%%%%%%%%
0648
0649         utils.helper.msg(msg.PROC1, 'constructing from X and Y values');
0650         a = a.fromVals(plist('XVALS', args{1}, 'YVALS', args{2}));
0651
0652     elseif ischar(args{1}) && isa(args{2}, 'plist')
0653         %%%%%%%%%%% ao('foo.txt', pl) %%%%%%%%%%%
0654
0655         utils.helper.msg(msg.PROC1, 'constructing from filename and plist');
0656         pl = combine(plist('filename', args{1}), args{2});
0657         a = a.fromFile(pl);
0658
0659     elseif isa(args{1}, 'ao') && isa(args{2}, 'ao')
0660         %%%%%%%%%%% ao(ao-object, ao-object) %%%%%%%%%%%
0661         % Do we have a list of AOs as input
0662         a = ao([args{1}, args{2}]);
0663
0664     elseif isa(args{1}, 'ao') && isa(args{2}, 'plist') && isempty(args{2}.params)
0665         % pass to copy constructor
0666         a = ao(args{1});
0667     elseif isa(args{1}, 'ao') && isa(args{2}, 'plist')
0668         %%%%%%%%%%% ao(ao-object, plist-object) %%%%%%%%%%%
0669         % This constructor is necessary for the function type.
0670         % Ignore the first AO and build a new AO from the plist.
0671
0672         a = ao(args{2});
0673     else
0674         error('### Unknown constructor with two inputs');
0675     end
0676
0677 otherwise
0678     %%%%%%%%%%% other inputs %%%%%%%%%%%
0679     %%%%%%%%%%% other inputs %%%%%%%%%%%
0680     %%%%%%%%%%% other inputs %%%%%%%%%%%
0681
0682     [aoi, invars, rest] = utils.helper.collect_objects(args, 'ao');
0683
0684     %%% Do we have a list of AOs as input
0685     if ~isempty(aoi) && isempty(rest)
0686         a = ao(aoi);
0687     else
0688         error('### Unknown number of arguments. ');
0689     end
0690 end
0691 end % End constructor
0692 end % End public methods
0693
0694
0695 %%%%%%%%%%% Methods (static) %%%%%%%%%%%
0696 %
0697 %%%%%%%%%%% Methods (static) %%%%%%%%%%%
0698 methods (Static=true)
0699     varargout = mdc1_input_noises(varargin)
0700     varargout = update_struct(varargin);
0701

```

```

0702 function out = SETS()
0703     out = {...
0704         'Default', ...
0705         'From XML File', 'From MAT File', 'From ASCII File',...
0706         'From Function', 'From Values',...
0707         'From Time-series Function', 'From Frequency-series Function', ...
0708         'From Window', 'From Waveform', 'From Polynomial', 'From Repository',...
0709         'From Plist', 'From Pzmodel'};
0710 end
0711
0712 function out = VEROUT()
0713     out = '$Id: ao.m,v 1.158 2008/09/07 18:11:06 hewitson Exp $';
0714 end
0715
0716 function ii = getInfo(varargin)
0717     ii = utils.helper.generic_getInfo(varargin{:}, 'ao');
0718 end
0719
0720 % Return the plist for a particular parameter set
0721 function out = getDefaultPlist(set)
0722
0723     % Otherwise we try to find a set for this constructor
0724     switch set
0725     case 'Default'
0726         out = plist();
0727
0728         %-----
0729         %--- Repository constructor
0730         %-----
0731     case 'From Repository'
0732         out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
0733
0734         %-----
0735         %--- Read from XML file
0736         %-----
0737     case 'From XML File'
0738         out = plist('filename', '');
0739
0740         %-----
0741         %--- Read from MAT file
0742         %-----
0743     case 'From MAT File'
0744         out = plist('filename', '');
0745
0746         %-----
0747         %--- Read from ASCII file
0748         %-----
0749     case 'From ASCII File'
0750         out = plist(...
0751             'filename', '', ...
0752             'type', 'tsdata', ...
0753             'columns', [1 2], ...
0754             'xunits', unit('s'), ...
0755             'yunits', unit('V'), ...
0756             't0', time(0), ...
0757             'comment_char', '', ...
0758             'use_fs', '', ...
0759             'Robust', 'yes');
0760
0761         %-----
0762         %--- Create from a function description
0763         %-----
0764     case 'From Function'
0765         out = plist('fcn', 'randn(100,1)');
0766
0767         %-----
0768         %--- Create from a set of values
0769         %-----
0770     case 'From Values'
0771         out = plist('vals', 1, 'N', 1);
0772
0773         %-----
0774         %--- Create from a time-series function
0775         %-----
0776     case 'From Time-series Function'
0777         out = plist('tsfcn', 't', 'fs', 10, 'nsecs', 1, 't0', time(0));
0778
0779         %-----
0780         %--- Create from frequency-series function
0781         %-----
0782     case 'From Frequency-series Function'
0783         out = plist('fsfcn', 'f', 'f1', 1e-9, 'f2', 5, 'nf', 1000, 'scale', 'log',
0784             'f', []);
0785
0786         %-----
0787         %--- Create from a window function

```

```

0787         %-----
0788     case 'From Window'
0789         out = plist('win', specwin('Hanning', 100));
0790
0791         %-----
0792         %--- Create from a set of polynomial coefficients
0793         %-----
0794     case 'From Polynomial'
0795         out = plist('polyval', [-1e-4 .02 -1 -1], 'Nsecs', 10, 'fs', 10, 't', []);
0796
0797         %-----
0798         %--- Create from a plist
0799         %-----
0800     case 'From Plist'
0801         out = plist('Plist', plist);
0802
0803         %-----
0804         %--- Create from a waveform description
0805         %-----
0806     case 'From Waveform'
0807         out = plist('waveform', 'sine wave', ...
0808             'A', 1, ...
0809             'f', 1.23, ...
0810             'phi', 0, ...
0811             'fs', 10, ...
0812             'nsecs', 10, ...
0813             't0', time(0));
0814
0815         %-----
0816         %--- Create from a set of polynomial coefficients
0817         %-----
0818     case 'From Pzmodel'
0819         out = plist('pzmodel', pzmodel, 'Nsecs', 0, 'fs', 0);
0820     otherwise
0821         out = plist();
0822     end
0823 end % End getDefaultPlist
0824 end
0825
0826
0827 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0828 %                                     Methods (public)                                     %
0829 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0830 methods
0831 % Getters
0832 val = xunits(varargin)
0833 val = yunits(varargin)
0834 val = fs(varargin)
0835 val = t0(varargin)
0836 val = x(varargin)
0837 val = y(varargin)
0838 end
0839
0840 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0841 %                                     Methods (static, private)                               %
0842 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0843 methods (Static=true, Access=private)
0844 % constructor functions
0845 % Spectral estimate function
0846 [yy, info] = welchscale(xx, win, fs, norm, inunits)
0847 varargout = welch(varargin)
0848 [x,M,isreal_x,y,Ly,win,winName,winParam,noverlap,k,L,options] =
0849 welchparse(x,esttype,varargin)
0849 [P,f] = computeperiodogram(x,win,nfft,esttype,varargin)
0850 [Xx,f] = computedFT(xin,nfft,varargin)
0851 varargout = xspec(varargin)
0852
0853 % Noise generator functions
0854 varargout = ngconv(varargin)
0855 varargout = ngsetup(varargin)
0856 varargout = ngsetup_vpa(varargin)
0857 varargout = nginit(varargin)
0858 varargout = ngprop(varargin)
0859 varargout = fq2fac(varargin)
0860 varargout = conv_noisegen(varargin)
0861 varargout = mchol(varargin)
0862
0863 % LPSD-type methods
0864 varargout = ltf_plan(varargin)
0865 varargout = mlpsd_mex(varargin)
0866 varargout = mlpsd_m(varargin)
0867 varargout = mltfe(varargin)
0868 varargout = findFsMax(varargin)
0869 varargout = findShortestVector(varargin)
0870 varargout = lxspect(varargin)
0871 end

```



```
0872
0873
0874 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0875 %                                     Methods (public) %
0876 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0877 methods
0878
0879 % Setters
0880 varargout = setXunits(varargin)
0881 varargout = setYunits(varargin)
0882 varargout = setT0(varargin)
0883 varargout = setFs(varargin)
0884 varargout = setXY(varargin)
0885 varargout = setY(varargin)
0886 varargout = setX(varargin)
0887 varargout = setDescription(varargin)
0888 varargout = setZ(varargin)
0889
0890
0891 % Other methods
0892 varargout = copy(varargin)
0893 h = md5(varargin)
0894 end % End public methods
0895
0896
0897 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0898 %                                     Methods (protected) %
0899 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0900 methods (Access = protected)
0901     obj = fromDataInMAT(obj, data, filename)
0902     obj = fromDatafile(obj, pli)
0903 end
0904
0905
0906 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0907 %                                     Methods (private) %
0908 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0909 methods (Access=private)
0910     % Constructors
0911     a = fromVals(a, pli)
0912     a = fromTSfcn(a, pli)
0913     a = fromWaveform(a, pli)
0914     a = fromFcn(a,pli)
0915     a = fromFSfcn(a,pli)
0916     a = fromSpecWin(a,pli)
0917     a = fromPolyval(a,pli)
0918     a = fromPzmodel(a,pli)
0919     % Others
0920     varargout = applymethod(varargin)
0921     varargout = applyoperator(varargin)
0922 end
0923
0924 end
```

plist

PURPOSE ^

PLIST Plist class object constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
PLIST Plist class object constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PLIST Parameter-List Object class object constructor.
             Create a plist object.

SUPER CLASSES: ltpda_uo < ltpda_obj

PROPERTIES:

  Inherit Properties (read only)
    name      - name of object
    created   - creation time (time-object)
    prov      - contains a instance of the provenance class.
    hist      - history of the object (history object)
    version   - cvs-version string.

  Protected Properties (read only)
    params    - list of param-objects

PLIST METHODS:

  Defined Abstract methods:
    char      - returns one character string which represents the object
    copy      - copies an object
    display   - displays an object
    string    - converts an object to a command string which will
               recreate the plist object.
    update_struct - updates a object structure to the current tbx-version

  Public methods:
    append    - append a param-object, plist-object or a key/value pair to
               the parameter list.
    combine    - combine multiple parameter lists (plist objects) into a
               single plist.
    find      - Returns the value corresponding to the first parameters in
               the list with search-key.
    isparam   - look for a given key in the parameter lists.
    nparams   - returns the number of param objects in the list.
    pset      - set or add a key/value pair or a param-object into
               the parameter list.
    pzmresp   - shadow function for the pzmodel/resp.
    remove    - remove a parameter from the parameter list.
    resp      - shadows miir/iirResp and pzmodel/resp.

CONSTRUCTORS:

    pl = plist()           - create an empty plist object.
    pl = plist(p)          - create a plist with elements p
                           where p is an array of param objects.
    pl = plist('key', val) - create a plist with the key/value pair
    pl = plist('key1', val1, ... - create a plist with more key/value pairs
                           'key2', 'val2')
    pl = plist('file.xml') - load a plist-object from xml-file
    pl = plist('file.mat') - load a plist-object from mat-file
    pl = plist(pl)         - copies the input plist.

PARAMETERS:

  If no recognised parameters are found in the input plist, the input
  plist is simply returned. This is the copy constructor.
```



```
'Hostname' - construct a plist by retrieving it from an LTPDA repository
             specified by the given hostname. Only those objects which
             are plists are returned.
Additional parameters:
'Database'  - The database name [default: 'ltpda']
'ID'        - A vector of object IDs.
'CID'       - Retrieve all plists from a particular
              collection.

M-FILE INFO: The following call returns an minfo object that contains
              information about the plist constructor:
              >> info = plist.getInfo
or            >> info = plist.getInfo('plist')

              You can get information about class methods by calling:
              >> info = plist.getInfo(method)
e.g. >> info = plist.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
              >> info = plist.getInfo(method, set)
e.g. >> info = plist.getInfo('plist', 'Default')

VERSION:     $Id: plist.m,v 1.51 2008/09/07 18:15:26 hewitson Exp $

HISTORY:     30-01-07 M Hewitson
              Creation

SEE ALSO:    ltpda_obj, ltpda_uo, param

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [append](#) APPEND append a param-object, plist-object or a key/value pair to the parameter list.
-  [char](#) CHAR convert a parameter list into a string.
-  [combine](#) COMBINE multiple parameter lists (plist objects) into a single plist.
-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [display](#) DISPLAY display plist object.
-  [find](#) FIND overloads find routine for a parameter list.
-  [fromFile](#) Construct a ltpda_ob from a file
-  [fromRepository](#) Construct a plist filter from a repository
-  [isparam](#) ISPARAM look for a given key in the parameter lists.
-  [nparams](#) NPARAMS returns the number of param objects in the list.
-  [plist](#) PLIST Plist class object constructor.
-  [pset](#) PSET set or add a key/value pair or a param-object into the parameter list.
-  [remove](#) REMOVE remove a parameter from the parameter list.
-  [resp](#) RESP shadows miir/iirResp and pzmodel/resp.
-  [string](#) STRING converts a plist object to a command string which will recreate the plist object.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [append](#) APPEND append a param-object, plist-object or a key/value pair to the parameter list.
-  [char](#) CHAR convert a parameter list into a string.
-  [combine](#) COMBINE multiple parameter lists (plist objects) into a single plist.
-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [display](#) DISPLAY display plist object.
-  [find](#) FIND overloads find routine for a parameter list.
-  [fromFile](#) Construct a ltpda_ob from a file
-  [isparam](#) ISPARAM look for a given key in the parameter lists.
-  [nparams](#) NPARAMS returns the number of param objects in the list.
-  [plist](#) PLIST Plist class object constructor.

-  **pset** PSET set or add a key/value pair or a param-object into the parameter list.
-  **remove** REMOVE remove a parameter from the parameter list.
-  **resp** RESP shadows miir/iirResp and pzmodel/resp.
-  **string** STRING converts a plist object to a command string which will recreate the plist object.

SUBFUNCTIONS [^]

-  **function** pl = plist(varargin)
-  **function** out = VEROUT()
-  **function** ii = getInfo(varargin)
-  **function** out = SETS()
-  **function** out = getDefaultPlist(set)

SOURCE CODE [^]

```

0001 % PLIST Plist class object constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PLIST Parameter-List Object class object constructor.
0005 %             Create a plist object.
0006 %
0007 % SUPER CLASSES: ltpda_uo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         name      - name of object
0013 %         created    - creation time (time-object)
0014 %         prov       - contains a instance of the provenance class.
0015 %         hist       - history of the object (history object)
0016 %         version    - cvs-version string.
0017 %
0018 %     Protected Properties (read only)
0019 %         params     - list of param-objects
0020 %
0021 % PLIST METHODS:
0022 %
0023 %     Defined Abstract methods:
0024 %         char        - returns one character string which represents the object
0025 %         copy        - copies an object
0026 %         display     - displays an object
0027 %         string      - converts an object to a command string which will
0028 %                       recreate the plist object.
0029 %         update_struct - updates a object structure to the current tbx-version
0030 %
0031 %     Public methods:
0032 %         append      - append a param-object, plist-object or a key/value pair to
0033 %                       the parameter list.
0034 %         combine      - combine multiple parameter lists (plist objects) into a
0035 %                       single plist.
0036 %         find         - Returns the value corresponding to the first parameters in
0037 %                       the list with search-key.
0038 %         isparam      - look for a given key in the parameter lists.
0039 %         nparams      - returns the number of param objects in the list.
0040 %         pset         - set or add a key/value pair or a param-object into
0041 %                       the parameter list.
0042 %         pzmodelresp - shadow function for the pzmodel/resp.
0043 %         remove       - remove a parameter from the parameter list.
0044 %         resp         - shadows miir/iirResp and pzmodel/resp.
0045 %
0046 % CONSTRUCTORS:
0047 %
0048 %         pl = plist()           - create an empty plist object.
0049 %         pl = plist(p)         - create a plist with elements p
0050 %                               where p is an array of param objects.
0051 %         pl = plist('key', val) - create a plist with the key/value pair
0052 %         pl = plist('key1', val1, ... - create a plist with more key/value pairs
0053 %                               'key2', 'val2')
0054 %         pl = plist('file.xml') - load a plist-object from xml-file
0055 %         pl = plist('file.mat') - load a plist-object from mat-file
0056 %         pl = plist(pl)         - copies the input plist.
0057 %
0058 % PARAMETERS:
0059 %
0060 %     If no recognised parameters are found in the input plist, the input
0061 %     plist is simply returned. This is the copy constructor.

```

```
0062 %
0063 % 'Hostname' - construct a plist by retrieving it from an LTPDA repository
0064 % specified by the given hostname. Only those objects which
0065 % are plists are returned.
0066 % Additional parameters:
0067 % 'Database' - The database name [default: 'ltpda']
0068 % 'ID' - A vector of object IDs.
0069 % 'CID' - Retrieve all plists from a particular
0070 % collection.
0071 %
0072 % M-FILE INFO: The following call returns an minfo object that contains
0073 % information about the plist constructor:
0074 % >> info = plist.getInfo
0075 % or >> info = plist.getInfo('plist')
0076 %
0077 % You can get information about class methods by calling:
0078 % >> info = plist.getInfo(method)
0079 % e.g. >> info = plist.getInfo('eq')
0080 %
0081 % You can also restrict the sets of parameters contained in
0082 % the minfo object by calling:
0083 % >> info = plist.getInfo(method, set)
0084 % e.g. >> info = plist.getInfo('plist', 'Default')
0085 %
0086 % VERSION: $Id: plist.m,v 1.51 2008/09/07 18:15:26 hewitson Exp $
0087 %
0088 % HISTORY: 30-01-07 M Hewitson
0089 % Creation
0090 %
0091 % SEE ALSO: ltpda_obj, ltpda_uo, param
0092 %
0093 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0094 %
0095 classdef plist < ltpda_uo
0096 %
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098 % Property definition %
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100
0101 %----- Public (read/write) Properties -----
0102 properties
0103 end
0104
0105 %----- Protected read-only Properties -----
0106 properties (SetAccess = protected)
0107 params = [];
0108 end
0109
0110 %----- Protected Properties -----
0111 properties (SetAccess = protected)
0112 version = '$Id: plist.m,v 1.51 2008/09/07 18:15:26 hewitson Exp $';
0113 end
0114
0115 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116 % Check property setting %
0117 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0118
0119 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0120 % Constructor %
0121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122
0123 methods
0124 function pl = plist(varargin)
0125
0126 import utils.const.*
0127 utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0128
0129 %%% Call superclass
0130 pl = pl@ltpda_uo(varargin{:});
0131
0132 %%% Set default values %%%
0133
0134 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0135 % no input %%%
0136 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0137
0138 %%% pl = plist() %%%
0139 % create default parameter list object
0140 if nargin == 0
0141 utils.helper.msg(msg.OPROC1, 'empty constructor');
0142 elseif nargin == 1
0143
0144 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0145 % one input %%%
0146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0147
```

```

0148
0149 %%%%%%%%%% p = param(plist) %%%%%%%%%%
0150 if isa(varargin{1}, 'plist')
0151     pli = varargin{1};
0152
0153 %%% if the plist is empty then return an empty plist object
0154 if isempty(varargin{1}.params)
0155     else
0156
0157         % Retrieve from repository?
0158         hostname = find(pli, 'hostname');
0159         if ~isempty(hostname)
0160             utils.helper.msg(msg.OPROC1, 'retrieving from repository %s', hostname);
0161             pl = pl.fromRepository(pli);
0162         else
0163             utils.helper.msg(msg.OPROC1, 'copying from %s', varargin{1}.name);
0164             pl = copy(varargin{1}, 1);
0165         end
0166     end
0167
0168 %%%%%%%%%% From file %%%%%%%%%%
0169 elseif ischar(varargin{1})
0170
0171     filename = varargin{1};
0172     utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
0173     pl = fromFile(pl, filename);
0174
0175 %     [path, name, ext] = fileparts(filename);
0176 %     switch ext
0177 %     case '.mat'
0178 %         pl = load(filename);
0179 %         pl = pl.a;
0180 %     case '.xml'
0181 %         root_node = xmlread(filename);
0182 %         pl = utils.helper.xmlread(root_node, 'plist');
0183 %     otherwise
0184 %         error('### Unknown file type.');
```

```

0234         utils.helper.msg(msg.OPROC1, 'constructing from key/value pairs');
0235
0236         %%%%%%%%%%% pl = plist('key1, val1, 'key2', 'val2' , ...) %%%%%%%%%%%
0237         param_list = [];
0238         argin = varargin;
0239         while length(argin) >= 2
0240             key = upper(argin{1});
0241             val = argin{2};
0242             argin = argin(3:end);
0243
0244             if ~isempty(pl.params)
0245                 found = any(strcmpi(key, {pl.params(:).key}));
0246                 if found
0247                     error('### Do not use the same key [%s] twice.\n### REMARK: The key is
not case sensitive.', key);
0248                 end
0249             end
0250             % add to list
0251             param_list = [param_list param(key, val)];
0252         end
0253         pl.params = param_list;
0254     else
0255         error('### Unknown number of constructor arguments.');
```

0256 end

0257 end % End constructor

0258 end

0259

0260

0261 %%%%%%%%%%% %

0262 % Methods (public) %

0263 %%%%%%%%%%% %

0264

0265 methods

0266 %% Define Abstract methods

0267 varargout = char(varargin);

0268 varargout = copy(varargin);

0269 varargout = display(varargin);

0270 varargout = string(varargin);

0271

0272 %% Define other methods

0273 varargout = append(varargin);

0274 varargout = combine(varargin);

0275 varargout = find(varargin);

0276 varargout = isparam(varargin);

0277 varargout = nparams(varargin);

0278 varargout = pset(varargin);

0279 % varargout = pzmresp(varargin);

0280 varargout = remove(varargin);

0281 varargout = resp(varargin);

0282 end

0283

0284 %%%%%%%%%%% %

0285 % Methods (protected) %

0286 %%%%%%%%%%% %

0287

0288 methods (Access = protected)

0289 varargout = fromRepository(varargin)

0290 end

0291

0292 %%%%%%%%%%% %

0293 % Methods (Static, Public) %

0294 %%%%%%%%%%% %

0295

0296 methods (Static = true)

0297

0298 varargout = update_struct(varargin);

0299

0300 function out = VEROUT()

0301 out = '\$Id: plist.m,v 1.51 2008/09/07 18:15:26 hewitson Exp \$';

0302 end

0303

0304 function ii = getInfo(varargin)

0305 ii = utils.helper.generic_getInfo(varargin{:}, 'plist');

0306 end

0307

0308 function out = SETS()

0309 out = {'Default', 'From Repository', 'From MAT File'};

0310 end

0311

0312 function out = getDefaultPlist(set)

0313 switch set

0314 case 'Default'

0315 out = plist();

0316 case 'From Repository'

0317 out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);

0318 case 'From MAT File'

```
0319         out = plist('filename', '');
0320     otherwise
0321         error('### Unknown set [%s] to get the default plist.', set);
0322     end
0323 end
0324
0325 end
0326
0327 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0328 %                               Methods (Static, Private)                               %
0329 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0330
0331 methods (Static = true, Access = private)
0332 end
0333
0334 end % Class
0335
0336
```

param

PURPOSE ^

PARAM Parameter object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
PARAM Parameter object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PARAM Parameter object class constructor.
             Create a parameter object.

SUPER CLASSES: ltpda_nuo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
        version - cvs-version string.

    Protected Properties (read only)
        key      - key of the key/value pair
        val      - value of the key/value pair

PARAM METHODS:

    Defined Abstract methods:
        char      - returns one character string which represents the object
        copy      - copies an object
        display   - displays an object
        update_struct - updates a object structure to the current tbx-version

    Public methods:
        mux       - concatenate param-objects into a vector.
        setKey    - set the 'key' property
        setVal    - set the 'val' property
        setKeyVal - set the key/value pair
        string    - writes a command string that can be used to recreate the object

CONSTRUCTORS:

    p = param();           - creates an empty parameter
    p = param(pl)         - creates a parameter from a
                           parameter list with the parameters:
                           - 'key' and 'val', or
                           - 'filename'
    p = param(filename)    - load a parameter from file
    p = param('name', val) - creates a numeric parameter
    p = param('name', 'val') - creates a string parameter

M-FILE INFO: The following call returns an minfo object that contains
              information about the param constructor:
                >> info = param.getInfo
              or  >> info = param.getInfo('param')

              You can get information about class methods by calling:
                >> info = param.getInfo(method)
              e.g. >> info = param.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
                >> info = param.getInfo(method, set)
              e.g. >> info = param.getInfo('param', 'Default')

VERSION: $Id: param.m,v 1.44 2008/09/03 16:35:22 hewitson Exp $

HISTORY: 19-05-2008 Diepholz
          Creation.
```


SEE ALSO: [ltpda_obj](#), [ltpda_nuo](#), [plist](#)

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [copy](#) COPY Make copy of param objects depending of the second input
-  [display](#) DISPLAY display a parameter
-  [mux](#) MUX concatenate params into a vector.
-  [param](#) PARAM Parameter object class constructor.
-  [setKey](#) SETKEY Set the property 'key'.
-  [setKeyVal](#) SETKEYVAL Set the properties 'key' and 'val'
-  [setVal](#) SETVAL Set the property 'val'.
-  [string](#) STRING writes a command string that can be used to recreate the input param object.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [copy](#) COPY Make copy of param objects depending of the second input
-  [param](#) PARAM Parameter object class constructor.

SUBFUNCTIONS ^

-  [function obj = set.key\(obj, val\)](#)
-  [function obj = param\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 % PARAM Parameter object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PARAM Parameter object class constructor.
0005 %             Create a parameter object.
0006 %
0007 % SUPER CLASSES: ltpda_nuo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         version - cvs-version string.
0013 %
0014 %     Protected Properties (read only)
0015 %         key     - key of the key/value pair
0016 %         val     - value of the key/value pair
0017 %
0018 % PARAM METHODS:
0019 %
0020 %     Defined Abstract methods:
0021 %         char     - returns one character string which represents the object
0022 %         copy     - copies an object
0023 %         display  - displays an object
0024 %         update_struct - updates a object structure to the current tbx-version
0025 %
0026 %     Public methods:
0027 %         mux      - concatenate param-objects into a vector.
0028 %         setKey   - set the 'key' property
0029 %         setVal   - set the 'val' property
0030 %         setKeyVal - set the key/value pair
```



```

0031 %         string - writes a command string that can be used to recreate the object
0032 %
0033 % CONSTRUCTORS:
0034 %
0035 %         p = param(); - creates an empty parameter
0036 %         p = param(pl) - creates a parameter from a
0037 %                        parameter list with the parameters:
0038 %                        - 'key' and 'val', or
0039 %                        - 'filename'
0040 %         p = param(filename) - load a parameter from file
0041 %         p = param('name', val) - creates a numeric parameter
0042 %         p = param('name', 'val') - creates a string parameter
0043 %
0044 % M-FILE INFO: The following call returns an minfo object that contains
0045 %               information about the param constructor:
0046 %               >> info = param.getInfo
0047 %               or >> info = param.getInfo('param')
0048 %
0049 %               You can get information about class methods by calling:
0050 %               >> info = param.getInfo(method)
0051 %               e.g. >> info = param.getInfo('eq')
0052 %
0053 %               You can also restrict the sets of parameters contained in
0054 %               the minfo object by calling:
0055 %               >> info = param.getInfo(method, set)
0056 %               e.g. >> info = param.getInfo('param', 'Default')
0057 %
0058 % VERSION: $Id: param.m,v 1.44 2008/09/03 16:35:22 hewitson Exp $
0059 %
0060 % HISTORY: 19-05-2008 Diepholz
0061 %           Creation.
0062 %
0063 % SEE ALSO: ltpda_obj, ltpda_nuo, plist
0064 %
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066 %
0067 classdef (Sealed = true) param < ltpda_nuo
0068 %
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070 %               Property definition %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %----- Public (read/write) Properties -----
0073 properties
0074 end
0075
0076 %----- Protected read-only Properties -----
0077 properties (SetAccess = protected)
0078     key = '';
0079     val = [];
0080 end
0081
0082 %----- Protected Properties -----
0083 properties (SetAccess = protected)
0084     version = '$Id: param.m,v 1.44 2008/09/03 16:35:22 hewitson Exp $'
0085 end
0086
0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088 %               Check property setting %
0089 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090 %
0091 methods
0092     function obj = set.key(obj, val)
0093         if ischar(val)
0094             obj.key = upper(val);
0095         else
0096             error('### The value for the property ''key'' must be a string\n### but it is
0097 from the class %s', class(val));
0098         end
0099     end
0100 end
0101
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103 %               Constructor %
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %
0106 methods
0107     function obj = param(varargin)
0108
0109         %% Call superclass
0110         obj = obj@ltpda_nuo(varargin{:});
0111
0112         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113         Set default values %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114
0115         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116         no input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
0116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0117
0118 if nargin == 0
0119
0120     % Do nothing
0121
0122     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0123     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% one input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0124     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0125 elseif nargin == 1
0126
0127     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% obj = param(param) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0128     if isa(varargin{1}, 'param')
0129         obj = copy(varargin{1}, 1);
0130
0131     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% obj = param(plist) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0132     elseif isa(varargin{1}, 'plist')
0133
0134         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% obj = param(plist()) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0135         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% obj = param(plist('KEY', 'a', 'VAL', 1)) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0136         %%% is the plist is empty then return an empty param object
0137         if nparams(varargin{1}) == 0
0138
0139             else
0140                 pl = varargin{1};
0141                 key = find(pl, 'key');
0142                 val = find(pl, 'val');
0143                 if isempty(key)
0144                     error('### building a parameter from a plist requires one parameter in
the plist is called ''key''');
0145                 end
0146                 if isempty(val)
0147                     error('### building a parameter from a plist requires one parameter in
the plist is called ''val''');
0148                 end
0149
0150                 obj.key = key;
0151                 obj.val = val;
0152             end
0153
0154             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% obj = param(struct) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0155         elseif isstruct(varargin{1})
0156             %%% Set properties which are declared in this class
0157             obj.key = varargin{1}.key;
0158             obj.val = varargin{1}.val;
0159             obj.version = varargin{1}.version;
0160
0161             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% obj = param('filename.xml') %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0162         elseif ischar(varargin{1})
0163
0164             filename = varargin{1};
0165             [path, name, ext] = fileparts(filename);
0166             switch ext
0167                 case '.mat'
0168                     obj = load(filename);
0169                     obj = obj.a;
0170                 case '.xml'
0171                     root_node = xmlread(filename);
0172                     obj = utils.helper.xmlread(root_node, 'param');
0173                 otherwise
0174                     error('### Unknown file type.');
```

```
0200 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0201
0202 methods
0203     varargout = mux(varargin);
0204     varargout = string(varargin)
0205
0206 %%% Define Abstract methods
0207     varargout = char(varargin)
0208     varargout = copy(varargin)
0209     varargout = display(varargin)
0210
0211     varargout = setKey(varargin)
0212     varargout = setVal(varargin)
0213     varargout = setKeyVal(varargin)
0214 end
0215
0216 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0217 %                                     Methods (protected)                                     %
0218 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0219
0220 methods (Access = protected)
0221 end
0222
0223 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0224 %                                     Methods (Static, Public)                                     %
0225 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0226
0227 methods (Static = true)
0228
0229     varargout = update_struct(varargin);
0230
0231     function out = VEROUT()
0232         out = '$Id: param.m,v 1.44 2008/09/03 16:35:22 hewitson Exp $';
0233     end
0234
0235     function ii = getInfo(varargin)
0236         ii = utils.helper.generic_getInfo(varargin{:}, 'param');
0237     end
0238
0239     function out = SETS()
0240         out = {'Default'};
0241     end
0242
0243     function out = getDefaultPlist(set)
0244         switch set
0245             case 'Default'
0246                 out = plist();
0247             otherwise
0248                 error('### Unknown set [%s]', set);
0249         end
0250     end
0251
0252 end
0253
0254 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0255 %                                     Methods (Static, Private)                                     %
0256 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0257
0258 methods (Static = true, Access = private)
0259 end
0260
0261 end
0262
```

pzmodel

PURPOSE ^

PZMODEL constructor for pzmodel class.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
PZMODEL constructor for pzmodel class.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PZMODEL constructor for pzmodel class.

SUPER CLASSES: ltpda_uoh < ltpda_uo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
        name      - name of object
        created    - creation time (time-object)
        prov       - contains a instance of the provenance class.
        hist       - history of the object (history object)
        version    - cvs-version string.

    Protected Properties (read only)
        gain       - gain of the model
        poles      - pole-array of the model
        zeros      - zero-array of the model

PZMODEL Methods:

    Defined Abstract methods:
        char       - returns one character string which represents the object
        copy       - copies an object
        display    - displays an object
        string     - converts an object to a command string which will
                     recreate the plist object
        update_struct - updates a object structure to the current tbx-version

    Public methods:
        fngen      - creates an arbitrarily long time-series based on
                     the input pzmodel
        getlowerFreq - gets the frequency of the lowest pole or zero in
                     the model
        getupperFreq - gets the frequency of the highest pole or zero in
                     the model
        pzm2ab     - convert pzmodel to IIR filter coefficients using
                     bilinear transform
        resp       - returns the complex response of a pzmodel as
                     an Analysis Object
        tomfir     - approximates a pole/zero model with an FIR filter
        tomiir     - converts a pzmodel to an IIR filter using
```

a bilinear transform.

Protected methods:

```
setGain      - set the property 'gain',
setPoles     - set the property 'poles',
setZeros     - set the property 'zeros'
```

Private methods:

```
fromFile      - construct a pzmodel from a file
fromRepository - construct a pzmodel from a repository
fromPolesAndZeros - construct a pzmodel from poles and zeros
```

CONSTRUCTOR:

```
pzm = pzmodel()           - creates an empty pzmodel object
pzm = pzmodel(g, p, z)    - construct from gain, poles, zeros
pzm = pzmodel(g, p, z, 'name') - construct including name
pzm = pzmodel('foo.fil') - construct from LISO .fil file
pzm = pzmodel('foo.xml') - construct by loading the pzmodel from disk
pzm = pzmodel('foo.mat') - construct by loading the pzmodel from disk
pzm = pzmodel(pl)         - create a pzmodel object from the
                           description given in the parameter list.
```

Parameter sets for plist constructor (in order of priority):

From XML File

Construct a PZMODEL by loading it from an XML file.

```
'filename' - construct a PZMODEL from a filename.
Example: plist('filename', 'pzm1.xml')
[default: empty string]
```

From MAT File

Construct a PZMODEL by loading it from a MAT file.

```
'filename' - construct a PZMODEL from a filename.
Example: plist('filename', 'pzm1.mat')
[default: empty string]
```

From LISO File

Construct a PZMODEL by loading it from a LISO file.

```
'filename' - construct a PZMODEL from a filename.
Example: plist('filename', 'pzm1.fil')
[default: empty string]
```

From Repository

Construct a PZMODEL by retrieving it from an LTPDA repository.

```
'Hostname' - the repository hostname. Only those objects which
are PZMODELS are returned.
[default: 'localhost'];
```

Additional parameters:

```
'Database' - The database name [default: 'ltpda']
'ID'       - A vector of object IDs. [default: []]
'CID'      - Retrieve all pzmodel objects from a particular
collection.
```

From Poles/Zeros

```
-----
Construct a PZMODEL from poles and zeros.

'Gain'   - model gain [default: 1]
'Poles'  - vector of pole objects [default: empty pole]
'Zeros'  - vector of zero objects [default: empty zero]
'name'   - name of model [default: 'None']

Example: plist('name', 'lp_filter', 'poles', [pole(0.1) pole(1,100)], ...
         'gain', 2);

From Plist
-----

'Plist'   - construct from a plist. The value passed should be a plist
            object.
            [default: empty plist]

EXAMPLE 1: Still to be written ...

M-FILE INFO: The following call returns an minfo object that contains
              information about the pzmodel constructor:
              >> info = pzmodel.getInfo
or            >> info = pzmodel.getInfo('pzmodel')

              You can get information about class methods by calling:
              >> info = pzmodel.getInfo(method)
e.g. >> info = pzmodel.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
              >> info = pzmodel.getInfo(method, set)
e.g. >> info = pzmodel.getInfo('pzmodel', 'From Pzmodel')

VERSION:    $Id: pzmodel.m,v 1.60 2008/09/07 10:48:46 hewitson Exp $

HISTORY:    03-04-2007 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abcascade](#) ABCASCADE Cascade two filters together to get a new filter.
-  [char](#) CHAR convert a pzmodel object into a string.
-  [copy](#) COPY Make copy of pzmodel objects depending of the second input
-  [display](#) DISPLAY overloads display functionality for pzmodel objects.
-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input pzmodel.
-  [fromLISO](#) FROMLISO Construct a pzmodel from a LISO file
-  [fromPolesAndZeros](#)
%%%
-  [getlowerFreq](#) GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
-  [getupperFreq](#) GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
-  [pzm2ab](#) PZM2AB convert pzmodel to IIR filter coefficients using bilinear transform.
-  [pzmodel](#) PZMODEL constructor for pzmodel class.

-  [resp](#) RESP returns the complex response of a pzmodel as an Analysis Object.
-  [setGain](#) SETGAIN Set the property 'gain'
-  [setPoles](#) SETPOLES Set the property 'poles'
-  [setZeros](#) SETZEROS Set the property 'zeros'
-  [string](#) STRING writes a command string that can be used to recreate the input pzmodel object.
-  [tomfir](#) TOMFIR approximates a pole/zero model with an FIR filter.
-  [tomiir](#) TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [copy](#) COPY Make copy of pzmodel objects depending of the second input
-  [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS ^

-  [function obj = set.gain\(obj, val\)](#)
-  [function obj = set.poles\(obj, val\)](#)
-  [function obj = set.zeros\(obj, val\)](#)
-  [function obj = pzmodel\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 % PZMODEL constructor for pzmodel class.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PZMODEL constructor for pzmodel class.
0005 %
0006 % SUPER CLASSES: ltpda_uoh < ltpda_uo < ltpda_obj
0007 %
0008 % PROPERTIES:
0009 %
0010 %     Inherit Properties (read only)
0011 %         name      - name of object
0012 %         created    - creation time (time-object)
0013 %         prov       - contains a instance of the provenance class.
0014 %         hist       - history of the object (history object)
0015 %         version    - cvs-version string.
0016 %
0017 %     Protected Properties (read only)
0018 %         gain       - gain of the model
0019 %         poles      - pole-array of the model
0020 %         zeros      - zero-array of the model
0021 %
0022 % PZMODEL Methods:
0023 %
0024 %     Defined Abstract methods:
```

Description of pzmodel

```

0025 % char - returns one character string which represents the object
0026 % copy - copies an object
0027 % display - displays an object
0028 % string - converts an object to a command string which will
0029 % recreate the plist object
0030 % update_struct - updates a object structure to the current tbx-version
0031 %
0032 % Public methods:
0033 % fngen - creates an arbitrarily long time-series based on
0034 % the input pzmodel
0035 % getlowerFreq - gets the frequency of the lowest pole or zero in
0036 % the model
0037 % getupperFreq - gets the frequency of the highest pole or zero in
0038 % the model
0039 % pzm2ab - convert pzmodel to IIR filter coefficients using
0040 % bilinear transform
0041 % resp - returns the complex response of a pzmodel as
0042 % an Analysis Object
0043 % tomfir - approximates a pole/zero model with an FIR filter
0044 % tomiir - converts a pzmodel to an IIR filter using
0045 % a bilinear transform.
0046 %
0047 % Protected methods:
0048 % setGain - set the property 'gain'
0049 % setPoles - set the property 'poles'
0050 % setZeros - set the property 'zeros'
0051 %
0052 % Private methods:
0053 % fromFile - construct a pzmodel from a file
0054 % fromRepository - construct a pzmodel from a repository
0055 % fromPolesAndZeros - construct a pzmodel from poles and zeros
0056 %
0057 % CONSTRUCTOR:
0058 %
0059 % pzm = pzmodel() - creates an empty pzmodel object
0060 % pzm = pzmodel(g, p, z) - construct from gain, poles, zeros
0061 % pzm = pzmodel(g, p, z, 'name') - construct including name
0062 % pzm = pzmodel('foo.fil') - construct from LISO .fil file
0063 % pzm = pzmodel('foo.xml') - construct by loading the pzmodel from disk
0064 % pzm = pzmodel('foo.mat') - construct by loading the pzmodel from disk
0065 % pzm = pzmodel(pl) - create a pzmodel object from the
0066 % description given in the parameter list.
0067 %
0068 % Parameter sets for plist constructor (in order of priority):
0069 %
0070 % From XML File
0071 % -----
0072 %
0073 % Construct a PZMODEL by loading it from an XML file.
0074 %
0075 % 'filename' - construct a PZMODEL from a filename.
0076 % Example: plist('filename', 'pzm1.xml')
0077 % [default: empty string]
0078 %
0079 % From MAT File
0080 % -----
0081 %
0082 % Construct a PZMODEL by loading it from a MAT file.
0083 %
0084 % 'filename' - construct a PZMODEL from a filename.
0085 % Example: plist('filename', 'pzm1.mat')
0086 % [default: empty string]
0087 %
0088 % From LISO File
0089 % -----
0090 %
0091 % Construct a PZMODEL by loading it from a LISO file.
0092 %
0093 % 'filename' - construct a PZMODEL from a filename.

```



```

0094 %           Example: plist('filename', 'pzm1.fil')
0095 %           [default: empty string]
0096 %
0097 % From Repository
0098 % -----
0099 %
0100 %   Construct a PZMODEL by retrieving it from an LTPDA repository.
0101 %
0102 %   'Hostname' - the repository hostname. Only those objects which
0103 %               are PZMODELS are returned.
0104 %               [default: 'localhost'];
0105 %
0106 %   Additional parameters:
0107 %
0108 %       'Database' - The database name [default: 'ltpda']
0109 %       'ID'       - A vector of object IDs. [default: []]
0110 %       'CID'      - Retrieve all pzmodel objects from a particular
0111 %                   collection.
0112 %
0113 % From Poles/Zeros
0114 % -----
0115 %
0116 %   Construct a PZMODEL from poles and zeros.
0117 %
0118 %   'Gain' - model gain [default: 1]
0119 %   'Poles' - vector of pole objects [default: empty pole]
0120 %   'Zeros' - vector of zero objects [default: empty zero]
0121 %   'name'  - name of model [default: 'None']
0122 %
0123 %           Example: plist('name', 'lp_filter', 'poles', [pole(0.1) pole(1,100)], ...
0124 %                       'gain', 2);
0125 %
0126 % From Plist
0127 % -----
0128 %
0129 %   'Plist' - construct from a plist. The value passed should be a plist
0130 %             object.
0131 %             [default: empty plist]
0132 %
0133 % EXAMPLE 1: Still to be written ...
0134 %
0135 % M-FILE INFO: The following call returns an minfo object that contains
0136 %               information about the pzmodel constructor:
0137 %               >> info = pzmodel.getInfo
0138 %               or  >> info = pzmodel.getInfo('pzmodel')
0139 %
0140 %               You can get information about class methods by calling:
0141 %               >> info = pzmodel.getInfo(method)
0142 %               e.g. >> info = pzmodel.getInfo('eq')
0143 %
0144 %               You can also restrict the sets of parameters contained in
0145 %               the minfo object by calling:
0146 %               >> info = pzmodel.getInfo(method, set)
0147 %               e.g. >> info = pzmodel.getInfo('pzmodel', 'From Pzmodel')
0148 %
0149 % VERSION:      $Id: pzmodel.m,v 1.60 2008/09/07 10:48:46 hewitson Exp $
0150 %
0151 % HISTORY:      03-04-2007 M Hewitson
0152 %               Creation
0153 %
0154 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0155 %
0156 % classdef pzmodel < ltpda_uoh
0157 %
0158 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0159 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Property definition %
0160 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0161 %
0162 % ----- Public (read/write) Properties -----

```

```

0163 properties
0164 end
0165
0166 %----- Protected read-only Properties -----
0167 properties (SetAccess = protected)
0168     gain    = NaN;
0169     poles   = pz;
0170     zeros   = pz;
0171     version = '$Id: pzmodel.m,v 1.60 2008/09/07 10:48:46 hewitson Exp $';
0172 end
0173
0174 %----- Private Properties -----
0175 properties (GetAccess = protected, SetAccess = protected)
0176 end
0177
0178 %----- Abstract Properties -----
0179 properties (Abstract = true, SetAccess = protected)
0180 end
0181
0182 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0183 %                               Check property setting                               %
0184 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0185
0186 methods
0187     function obj = set_gain(obj, val)
0188         if ~isnumeric(val) || isempty(val) || ~isreal(val)
0189             error('### The value for the property ''gain'' must be a real number');
0190         end
0191         obj.gain = val;
0192     end
0193     function obj = set_poles(obj, val)
0194         if ~isa(val, 'pz') && ~isempty(val)
0195             error('### The value for the property ''poles'' must be an array of pz objects.');
```

```

0232         return
0233     end
0234
0235     switch nargin
0236     case 0
0237         utils.helper.msg(msg.OPROC1, 'empty constructor');
0238         obj.addHistory(pzmodel.getInfo('pzmodel', 'None'), [], [], []);
0239         %%%%%%%%%%%
0240         %%%%%%%%%%% Zero inputs %%%%%%%%%%%
0241         %%%%%%%%%%%
0242
0243     case 1
0244         %%%%%%%%%%%
0245         %%%%%%%%%%% One inputs %%%%%%%%%%%
0246         %%%%%%%%%%%
0247
0248         if ischar(varargin{1})
0249             %===== Filename
0250             %%%%%%%%%%% pzm = pzmodel('foo.mat') %%%%%%%%%%%
0251             %%%%%%%%%%% pzm = pzmodel('foo.xml') %%%%%%%%%%%
0252             %%%%%%%%%%% pzm = pzmodel('foo.liso') %%%%%%%%%%%
0253             utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
0254             obj = fromFile(obj, varargin{1});
0255
0256         elseif isstruct(varargin{1})
0257             %===== Struct
0258             %%%%%%%%%%% pzm = pzmodel(struct) %%%%%%%%%%%
0259
0260             %%% Set properties which are declared in this class
0261             utils.helper.msg(msg.OPROC1, 'constructing from struct');
0262             pzm_struct = varargin{1};
0263
0264             obj.gain = pzm_struct.gain;
0265             obj.poles = utils.helper.struct2obj(pzm_struct.poles, 'pz');
0266             obj.zeros = utils.helper.struct2obj(pzm_struct.zeros, 'pz');
0267             obj.version = varargin{1}.version;
0268
0269         elseif isa(varargin{1}, 'plist')
0270             %===== Plist
0271             %%%%%%%%%%% pzm = pzmodel(plist) %%%%%%%%%%%
0272             pl = varargin{1};
0273
0274             % Selection of construction method
0275             if pl.isparam('filename')
0276                 utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
0277                 % construct from file
0278                 obj = fromFile(obj, pl);
0279             elseif pl.isparam('hostname') || pl.isparam('conn')
0280                 utils.helper.msg(msg.OPROC1, 'constructing from repository %s', pl.find('hostname'));
0281                 % do hostname constructor
0282                 obj = obj.fromRepository(pl);
0283             elseif pl.isparam('gain') || pl.isparam('poles') || pl.isparam('zeros')
0284                 utils.helper.msg(msg.OPROC1, 'constructing from poles and zeros');
0285                 % Construct from poles and zeros
0286                 obj = fromPolesAndZeros(obj, pl);
0287             elseif pl.isparam('ipl')
0288                 ipl = find(pl, 'iplist');
0289                 % if the plist is empty, we return an empty PZMODEL
0290                 if ipl.nparams == 0
0291                     else
0292                         % do plist constructor
0293                         obj = pzmodel(ipl);
0294                     end
0295                 else
0296                     %%% if the plist is empty then return empty PZMODEL
0297                     if nparams(pl) == 0
0298                         else
0299                             error('### Unknown PZMODEL constructor method.');
```

```
0301         end
0302     else
0303         error('### Unknown single argument constructor.');
```

0304 end

0305

0306 case 2

0307 %%%%%%%%%%% Two inputs %%%%%%%%%%%

0308 %%%%%%%%%%% Two inputs %%%%%%%%%%%

0309 %%%%%%%%%%% Two inputs %%%%%%%%%%%

0310 if isa(varargin{1}, 'database') && isnumeric(varargin{2})

0311 %%%%%%%%%%% f = pzmodel(<database-object>, [IDs]) %%%%%%%%%%%

0312 % pzmodel(<database-object>, [IDs])

0313 utils.helper.msg(msg.OPROC1, 'retrieve from repository');

0314 obj = obj.fromRepository(plist('conn', varargin{1}, 'id', varargin{2}));

0315 elseif isa(varargin{1}, 'pzmodel') && isa(varargin{2}, 'plist') && isempty(varargin{2}.params)

0316 % pass to copy constructor

0317 obj = pzmodel(varargin{1});

0318 else

0319 error('### Unknown 2 argument constructor.');

0320 end

0321 end

0322

0323 case 3

0324 %%%%%%%%%%% Three inputs %%%%%%%%%%%

0325 %%%%%%%%%%% Three inputs %%%%%%%%%%%

0326 %%%%%%%%%%% Three inputs %%%%%%%%%%%

0327

0328 %%%%%%%%%%% pzm = pzmodel(gain, poles, zeros) %%%%%%%%%%%

0329 utils.helper.msg(msg.OPROC1, 'constructing from poles and zeros');

0330 pl = plist('gain', varargin{1}, 'poles', varargin{2}, 'zeros', varargin{3});

0331 obj = fromPolesAndZeros(obj, pl);

0332 case 4

0333 %%%%%%%%%%% Four inputs %%%%%%%%%%%

0334 %%%%%%%%%%% Four inputs %%%%%%%%%%%

0335 %%%%%%%%%%% Four inputs %%%%%%%%%%%

0336

0337 %%%%%%%%%%% pzm = pzmodel(gain, poles, zeros, name) %%%%%%%%%%%

0338 utils.helper.msg(msg.OPROC1, 'constructing from poles and zeros');

0339 pl = plist('gain', varargin{1}, 'poles', varargin{2}, 'zeros', varargin{3}, 'name', varargin{4});

0340 obj = fromPolesAndZeros(obj, pl);

0341

0342 otherwise

0343 [pzs, invars, rest] = utils.helper.collect_objects(args, 'pzmodel');

0344

0345 %%% Do we have a list of AOs as input

0346 if ~isempty(pzs) && isempty(rest)

0347 obj = pzmodel(pzs);

0348 else

0349 error('### Unknown number of arguments.');

0350 end

0351 end

0352

0353 end % End constructor

0354

0355 end

0356

0357 %%%%%%%%%%% Methods (protected) %%%%%%%%%%%

0358 %

0359 %%%%%%%%%%% Methods (protected) %%%%%%%%%%%

0360 methods (Access = protected)

0361 obj = setGain(obj, val)

0362 obj = setPoles(obj, val)

0363 obj = setZeros(obj, val)

0364 end

0365

0366 %%%%%%%%%%% Methods (static) %%%%%%%%%%%

0367 %

0368 %%%%%%%%%%% Methods (static) %%%%%%%%%%%

0369 methods (Static)

```

0370
0371     varargout = update_struct(varargin);
0372
0373     function out = VEROUT()
0374         out = '$Id: pzmodel.m,v 1.60 2008/09/07 10:48:46 hewitson Exp $';
0375     end
0376
0377     function ii = getInfo(varargin)
0378         ii = utils.helper.generic_getInfo(varargin{:}, 'pzmodel');
0379     end
0380
0381     function out = SETS()
0382         out = {...
0383             'Default', ...
0384             'From XML File', ...
0385             'From MAT File', ...
0386             'From LISO File', ...
0387             'From Repository', ...
0388             'From Poles/Zeros', ...
0389             'From Plist'};
0390     end
0391
0392     function out = getDefaultPlist(set)
0393         switch set
0394             case 'Default'
0395                 out = plist();
0396                 %--- Pole/zero constructor
0397             case 'From Poles/Zeros'
0398                 out = plist('gain', 1, ...
0399                     'poles', [], ...
0400                     'zeros', [], ...
0401                     'name', 'None');
0402                 %--- Repository constructor
0403             case 'From Repository'
0404                 out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
0405                 %--- Read from XML file
0406             case 'From XML File'
0407                 out = plist('filename', '');
0408                 %--- Read from MAT file
0409             case 'From MAT File'
0410                 out = plist('filename', '');
0411                 %--- Create from LISO file
0412             case 'From LISO File'
0413                 out = plist('filename', '');
0414                 %--- Create from a plist
0415             case 'From Plist'
0416                 out = plist('Plist', []);
0417             otherwise
0418                 out = plist();
0419         end
0420     end % function out = getDefaultPlist(varargin)
0421
0422     end % End static methods
0423
0424     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0425     %                               Methods (static, private) %
0426     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0427
0428     methods (Static, Access=private)
0429
0430         [ao,bo] = abccascade(a1,b1,a2,b2)
0431
0432     end % End static, private methods
0433
0434     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0435     %                               Methods (public) %
0436     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0437     methods
0438         varargout = char(varargin)

```

```
0439     varargout = display(varargin)
0440     varargout = string(varargin)
0441     varargout = copy(varargin)
0442
0443     f = getlowerFreq(varargin)
0444     f = getupperFreq(varargin)
0445     f = tomiir(varargin)
0446     f = tomfir(varargin)
0447     varargout = pzm2ab(varargin)
0448     varargout = fngen(varargin)
0449     varargout = resp(varargin)
0450 end
0451
0452 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0453 %                               Methods (protected)                               %
0454 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0455 methods (Access = protected)
0456     pzm = fromLISO(pzm, pli)
0457 end
0458
0459 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0460 %                               Methods (private)                               %
0461 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0462 methods (Access = private)
0463     % Constructors
0464     varargout = fromPolesAndZeros(varargin)
0465
0466     % Converters
0467     [a,b] = cp2iir(p, fs)
0468     [a,b] = rp2iir(p, fs)
0469     [a,b] = cz2iir(z, fs)
0470     [a,b] = rz2iir(z, fs)
0471 end
0472
0473 end % End classdef
0474
```

downsample

PURPOSE ^

DOWNSAMPLE AOs containing time-series data.

SYNOPSIS ^

function varargout = downsample(varargin)

DESCRIPTION ^

```
DOWNSAMPLE AOs containing time-series data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DOWNSAMPLE AOs containing time-series data.

CALL:      b = downsample(a, pl)      - use plist to get parameters
          b = downsample(a1, a2, pl) - downsample both a1 and a2; b is then a 2x1
                                   vector.

PARAMETERS: 'factor' - decimation factor [default: 1]
          'offset'  - sample offset for decimation [default: 0]

EXAMPLES:  1) downsample x4; offset is set to default of 0
           >> p = plist('factor',4);
           >> b = downsample(a, p);

           2) downsample x2 with 1 sample offset
           >> p = plist('factor',2,'offset',1);
           >> b = downsample(a,p);

M-FILE INFO: Get information about this methods by calling
           >> ao.getInfo('downsample')

           Get information about a specified set-plist by calling:
           >> ao.getInfo('downsample', 'None')

VERSION:    $Id: downsample.m,v 1.15 2008/09/05 11:05:29 ingo Exp $

HISTORY:    14-05-07 M Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % DOWNSAMPLE AOs containing time-series data.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DOWNSAMPLE AOs containing time-series data.
0005 %
0006 % CALL:          b = downsample(a, pl)          - use plist to get parameters
0007 %               b = downsample(a1, a2, pl) - downsample both a1 and a2; b is then a 2x1
0008 %                               vector.
0009 %
0010 % PARAMETERS: 'factor' - decimation factor [default: 1]
0011 %             'offset' - sample offset for decimation [default: 0]
0012 %
0013 % EXAMPLES:  1) downsample x4; offset is set to default of 0
0014 %
0015 %             >> p = plist('factor',4);
0016 %             >> b = downsample(a, p);
0017 %
0018 %             2) downsample x2 with 1 sample offset
0019 %
0020 %             >> p = plist('factor',2,'offset',1);
0021 %             >> b = downsample(a,p);
0022 %
0023 % M-FILE INFO: Get information about this methods by calling
0024 %             >> ao.getInfo('downsample')
0025 %
0026 %             Get information about a specified set-plist by calling:
0027 %             >> ao.getInfo('downsample', 'None')
0028 %
0029 % VERSION:      $Id: downsample.m,v 1.15 2008/09/05 11:05:29 ingo Exp $
0030 %
0031 % HISTORY:      14-05-07 M Hewitson
0032 %               Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035
0036 function varargout = downsample(varargin)
0037
0038 % Check if this is a call for parameters
0039 if utils.helper.isinfocall(varargin{:})
0040     varargout{1} = getInfo(varargin{3});
0041     return
0042 end
0043
0044 import utils.const.*
0045 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0046
0047 % Collect input variable names
0048 in_names = cell(size(varargin));
0049 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0050
0051 % Collect all AOs
0052 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0053 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0054
0055 % Decide on a deep copy or a modify
0056 bs = copy(as, nargin);
0057
0058 % Combine plists
0059 pl = combine(pl, getDefaultPlist);
0060
0061 % Check for input plist
0062 if isempty(pl)
0063     error('### Please give a plist with a parameter ''Factor''.');
0064 end
0065
0066 % Get parameters from plist
0067 offset = find(pl, 'offset');
0068 factor = find(pl, 'factor');
0069
0070 % Checking downsampling value is valid
0071 if isempty(factor)
0072     error('### Please give a plist with a parameter ''Factor''.');
0073 end
0074
0075 % Checking downsampling value is integer
0076 if rem(factor, floor(factor)) ~= 0
0077     warning('!!! Downsample factor should be an integer. Rounding. !!!');
0078     factor = round(factor);
0079 end
0080
0081 % Checking sample offset value
0082 if isempty(offset)
0083     warning('!!! No offset specified; using default of 0 samples !!!');
0084     offset = 0;
0085 end
0086

```



```

0087 % Loop over input AOs
0088 for j=1:numel(bs)
0089     if isa(bs(j).data, 'tsdata')
0090         % get samples
0091         ss = 1+offset;
0092         samples = ss:factor:length(bs(j).data.y);
0093         % select samples
0094         bs(j).setXY(bs(j).data.getX(samples), bs(j).data.getY(samples), 'internal');
0095         % Set new sample rate
0096         bs(j).setFs(bs(j).data.fs/factor, 'internal');
0097         % drop X vector again if we can
0098         bs(j).data.collapseX;
0099     else
0100         error('### I can only downsample time-series AOs.');
```

```

0101     end
0102
0103     % Add history
0104     bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0105     % set name
0106     bs(j).setName(sprintf('downsample(%s)', ao_invars{j}), 'internal');
0107 end
0108
0109 % Set output
0110 if nargout > 0
0111     varargout{1} = bs;
0112 end
0113 end
0114
0115 %-----
0116 % Get Info Object
0117 %-----
0118 function ii = getInfo(varargin)
0119     if nargin == 1 && strcmpi(varargin{1}, 'None')
0120         sets = {};
0121         pl = [];
0122     else
0123         sets = {'Default'};
0124         pl = getDefaultPlist;
0125     end
0126     % Build info object
0127     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: downsample.m,v
1.15 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0128 end
0129
0130 %-----
0131 % Get Default Plist
0132 %-----
0133 function pl = getDefaultPlist()
0134     pl = plist('factor', 1, 'offset', 0);
0135 end
0136
0137

```

upsample

PURPOSE ^

UPSAMPLE overloads upsample function for AOs.

SYNOPSIS ^

function varargout = upsample(varargin)

DESCRIPTION ^

```
UPSAMPLE overloads upsample function for AOs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: UPSAMPLE overloads upsample function for AOs.

        A signal at sample rate fs is upsampled by inserting N-1 zeros between the
        input samples.

CALL:      b = upsample(a, pl)

Parameters: 'N'      - specify the desired upsample rate.
            'phase' - specify an initial phase in range [0, N-1].

M-FILE INFO: Get information about this methods by calling
            >> ao.getInfo('upsample')

            Get information about a specified set-plist by calling:
            >> ao.getInfo('upsample', 'None')

VERSION:    $Id: upsample.m,v 1.12 2008/09/05 11:05:29 ingo Exp $

HISTORY: 19-03-07 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [phase](#) PHASE overloads the ltpda_phase operator for Analysis objects.
-  [upsample](#) UPSAMPLE overloads upsample function for AOs.

This function is called by:

-  [upsample](#) UPSAMPLE overloads upsample function for AOs.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % UPSAMPLE overloads upsample function for AOs.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: UPSAMPLE overloads upsample function for AOs.
0005 %
0006 %           A signal at sample rate fs is upsampled by inserting N-1 zeros between
the
0007 %           input samples.
0008 %
0009 % CALL:      b = upsample(a, pl)
0010 %
0011 % Parameters: 'N'      - specify the desired upsample rate.
0012 %           'phase' - specify an initial phase in range [0, N-1].
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %           >> ao.getInfo('upsample')
0016 %
0017 %           Get information about a specified set-plist by calling:
0018 %           >> ao.getInfo('upsample', 'None')
0019 %
0020 % VERSION:    $Id: upsample.m,v 1.12 2008/09/05 11:05:29 ingo Exp $
0021 %
0022 % HISTORY: 19-03-07 M Hewitson
0023 %           Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026 %
0027 function varargout = upsample(varargin)
0028 %
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034 %
0035 import utils.const.*
0036 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0037 %
0038 % Collect input variable names
0039 in_names = cell(size(varargin));
0040 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0041 %
0042 % Collect all AOs and plists
0043 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0044 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0045 %
0046 % Decide on a deep copy or a modify
0047 bs = copy(as, nargout);
0048 %
0049 % combine plists
0050 pl = combine(pl, getDefaultPlist());
0051 %
0052 % Get output sample rate
0053 Nup = find(pl, 'N');
0054 if isempty(Nup)
0055     error('### Please give a plist with a parameter 'N'.');
0056 end
0057 % Get initial phase
0058 phase = find(pl, 'phase');
0059 if isempty(phase)
0060     phase = 0;
0061 end
0062 %
0063 % Loop over AOs
0064 for j=1:numel(bs)
0065     if ~isa(bs(j).data, 'tsdata')
0066         warning('!!! Upsample only works on tsdata objects. Skipping AO %s',
ao_invars{j});
0067         bs(j) = [];
0068     else
0069         % upsample y
0070         bs(j).setY(upsample(bs(j).data.y, floor(Nup), phase), 'internal');
0071         % Correct fs
0072         bs(j).setFs(floor(Nup)*bs(j).data.fs, 'internal');
0073         % Set name
0074         bs(j).setName(sprintf('upsample(%s)', ao_invars{j}), 'internal');
0075         % Add history
0076         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0077     end
0078 end
0079 %
0080 % Set outputs
0081 if nargout > 0
0082     varargout{1} = bs;
0083 end
0084 end
0085 %
0086 %-----
0087 % Get Info Object

```

```
0088 %-----
0089 function ii = getInfo(varargin)
0090     if nargin == 1 && strcmpi(varargin{1}, 'None')
0091         sets = {};
0092         pl = [];
0093     else
0094         sets = {'Default'};
0095         pl = getDefaultPlist;
0096     end
0097     % Build info object
0098     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: upsample.m,v
1.12 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0099 end
0100
0101 %-----
0102 % Get Default Plist
0103 %-----
0104 function pl_default = getDefaultPlist()
0105     pl_default = plist('N', 1, 'phase', 0);
0106 end
0107 % END
0108
```

resample

PURPOSE ^

RESAMPLE overloads resample function for AOs.

SYNOPSIS ^

function varargout = resample(varargin)

DESCRIPTION ^

```
RESAMPLE overloads resample function for AOs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RESAMPLE overloads resample function for AOs.

CALL:      b = resample(a, pl)

Parameters: 'fsout' - specify the desired output sample rate

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('resample')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('resample', 'None')

VERSION:    $Id: resample.m,v 1.30 2008/09/05 11:05:29 ingo Exp $

HISTORY:    19-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [resample](#) RESAMPLE overloads resample function for AOs.
-  [t0](#) T0 Get the data property 't0'.

This function is called by:

-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.
-  [lxspect](#) LXSPEC performs log-scale cross-spectral analysis of various forms.
-  [resample](#) RESAMPLE overloads resample function for AOs.
-  [xspect](#) XSPEC performs cross-spectral analysis of various forms.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % RESAMPLE overloads resample function for AOs.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RESAMPLE overloads resample function for AOs.
0005 %
0006 % CALL:          b = resample(a, pl)
0007 %
0008 % Parameters: 'fsout' - specify the desired output sample rate
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %               >> ao.getInfo('resample')
0012 %
0013 %               Get information about a specified set-plist by calling:
0014 %               >> ao.getInfo('resample', 'None')
0015 %
0016 % VERSION:       $Id: resample.m,v 1.30 2008/09/05 11:05:29 ingo Exp $
0017 %
0018 % HISTORY:       19-03-07 M Hewitson
0019 %               Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 function varargout = resample(varargin)
0024
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038 % Collect all AOs and plists
0039 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0041
0042 % Decide on a deep copy or a modify
0043 bs = copy(as, nargout);
0044
0045 % Combine plists
0046 pl = combine(pl, getDefaultPlist);
0047
0048 if isempty(pl)
0049     error('### Please give a plist with a parameter ''fsout''.');
0050 end
0051
0052 % Get output sample rate
0053 fsout = find(pl, 'fsout');
0054 if isempty(fsout)
0055     error('### Please give a plist with a parameter ''fsout''.');
0056 end
0057
0058 % Get a filter if specified
0059 filt = find(pl, 'filter');
0060
0061 % Loop over AOs
0062 for j=1:numel(bs)
0063     if ~isa(bs(j).data, 'tsdata')
0064         warning('!!! Skipping non-tsdata AO: %s', ao_invars{j});
0065         bs(j) = [];
0066     else
0067         % Compute the resampling factors
0068         [P,Q] = utils.math.intfact(fsout,bs(j).data.fs);
0069         utils.helper.msg(msg.PROC1, 'resampling by %g/%g', P, Q);
0070
0071         % Check we have an evenly sampled data series
0072         [fs, t0, fitted] = tsdata.fitfs(bs(j).data.getX);
0073         if fitted
0074             error('### The AO %s is unevenly sampled. It can not be resampled this way.',
ao_invars{j});
0075         end
0076         % resample y
0077         if isempty(filt)
0078             bs(j).setY(resample(bs(j).data.getY, P, Q), 'internal');
0079         else
0080             [p,q] = rat( fsout/d.fs, 1e-12 );
0081             b = p*filt.a;
0082             bs(j).setY(resample(bs(j).data.getY, P, Q, b), 'internal');
0083         end
0084         % Set new sample rate
0085         bs(j).setFs(fsout, 'internal');

```

```

0086         % Set output AO name
0087         bs(j).setName(sprintf('resample(%s)', ao_invars{j}), 'internal');
0088         % Add history
0089         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0090     end
0091 end
0092
0093 % Set output
0094 if nargout > 0
0095     varargout{1} = bs;
0096 end
0097 end
0098
0099 %-----
0100 % Get Info Object
0101 %-----
0102 function ii = getInfo(varargin)
0103     if nargin == 1 && strcmpi(varargin{1}, 'None')
0104         sets = {};
0105         pl = [];
0106     else
0107         sets = {'Default'};
0108         pl = getDefaultPlist;
0109     end
0110     % Build info object
0111     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: resample.m,v
1.30 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0112 end
0113
0114 %-----
0115 % Get Default Plist
0116 %-----
0117 function pl_default = getDefaultPlist()
0118     pl_default = plist('fsout', []);
0119 end
0120 % END
0121

```

interp

PURPOSE ^

INTERP interpolate the values in the input AO(s) at new values.

SYNOPSIS ^

function varargout = interp(varargin)

DESCRIPTION ^

INTERP interpolate the values in the input AO(s) at new values.
%%%

DESCRIPTION: INTERP interpolate the values in the input AO(s) at new values specified by the input parameter list.

CALL: b = interp(a, pl)

INPUTS: a - input array of AOs
 pl - parameter list with the keys 'vertices' and 'method'

OUTPUTS: b - output array of AOs

PARAMETERS: 'vertices' - new set of vertices to interpolate on.
 'method' - one of:
 'nearest' - nearest neighbour
 'linear' - linear interpolation
 'spline' - spline interpolation [default]
 'cubic' - shape-preserving piecewise cubic interpolation

REMARKs: 1) Matrix cdata objects are not supported.
 2) If a time-series object is interpolated, the sample rate is adjusted to the best fit of the new data.

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('interp')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('interp', 'None')

VERSION: \$Id: interp.m,v 1.24 2008/09/05 11:05:29 ingo Exp \$

HISTORY: 05-06-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [t0](#) T0 Get the data property 't0'.

This function is called by:

-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.
-  [fixfs](#) FIXFS resamples the input time-series to have a fixed sample rate.
-  [interpmissing](#) INTERPMISSING interpolate missing samples in a time-series.

SUBFUNCTIONS [^]

 `function ii = getInfo(varargin)`
 `function pl_default = getDefaultPlist()`

SOURCE CODE [^]

```

0001 % INTERP interpolate the values in the input AO(s) at new values.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: INTERP interpolate the values in the input AO(s) at new values
0005 %               specified by the input parameter list.
0006 %
0007 % CALL:        b = interp(a, pl)
0008 %
0009 % INPUTS:      a - input array of AOs
0010 %              pl - parameter list with the keys 'vertices' and 'method'
0011 %
0012 % OUTPUTS:     b - output array of AOs
0013 %
0014 % PARAMETERS:  'vertices' - new set of vertices to interpolate on.
0015 %              'method'   - one of:
0016 %                  'nearest' - nearest neighbour
0017 %                  'linear'  - linear interpolation
0018 %                  'spline'  - spline interpolation [default]
0019 %                  'cubic'   - shape-preserving piecewise cubic interpolation
0020 %
0021 % REMARKS:     1) Matrix cdata objects are not supported.
0022 %              2) If a time-series object is interpolated, the sample rate
0023 %                  is adjusted to the best fit of the newdata.
0024 %
0025 % M-FILE INFO: Get information about this methods by calling
0026 %              >> ao.getInfo('interp')
0027 %
0028 %              Get information about a specified set-plist by calling:
0029 %              >> ao.getInfo('interp', 'None')
0030 %
0031 % VERSION:     $Id: interp.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
0032 %
0033 % HISTORY:     05-06-07 M Hewitson
0034 %              Creation
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037
0038 function varargout = interp(varargin)
0039
0040 % Check if this is a call for parameters
0041 if utils.helper.isinfocall(varargin{:})
0042     varargout{1} = getInfo(varargin{3});
0043     return
0044 end
0045
0046 import utils.const.*
0047 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0048
0049 % Collect input variable names
0050 in_names = cell(size(varargin));
0051 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0052
0053 % Collect all AOs and plists
0054 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0055 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0056
0057 % Decide on a deep copy or a modify
0058 bs = copy(as, nargout);
0059
0060 % Combine plists
0061 pl = combine(pl, getDefaultPlist);
0062
0063 % Get parameters
0064 vertices = find(pl, 'vertices');
0065 method = find(pl, 'method');
0066
0067 %-----
0068 % Loop over input AOs
0069 for j=1: numel(bs)
0070     %-----
0071     % Interpolate this vector
0072     if ~isa(bs(j).data, 'cdata')
0073         % for tsdata, fsdata and xydata objects
0074         bs(j).data.setXY(vertices, interp1(bs(j).data.getX, bs(j).data.getY, vertices,
```

```

method, 'extrap'));
0075     else
0076         % for cdata object
0077         bs(j).data.setY(interp1(bs(j).data.y,vertices, method, 'extrap'));
0078     end
0079
0080     % Adjust sample rate for tsdata
0081     if isa(bs(j).data, 'tsdata')
0082         utils.helper.msg(msg.PROC1, 'adjusting sample rate of new data to best fit');
0083         [fs, t0, fitted] = tsdata.fitfs(bs(j).data.getX);
0084         utils.helper.msg(msg.PROC2, 'got new sample rate of %g Hz', fs);
0085         bs(j).data.setFs(fs);
0086         if ~fitted
0087             bs(j).data.collapseX;
0088         end
0089     end
0090     % Add history
0091     bs(j).addHistory(getInfo, pl, ao_invars, bs(j).hist);
0092     % set name
0093     bs(j).setName(sprintf('%s(%s)', method, ao_invars{j}), 'internal');
0094 end
0095
0096 % Set outputs
0097 if nargout > 0
0098     varargout{1} = bs;
0099 end
0100 end
0101
0102 %-----
0103 % Get Info Object
0104 %-----
0105 function ii = getInfo(varargin)
0106     if nargin == 1 && strcmpi(varargin{1}, 'None')
0107         sets = {};
0108         pl = [];
0109     else
0110         sets = {'Default'};
0111         pl = getDefaultPlist;
0112     end
0113     % Build info object
0114     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: interp.m,v 1.24
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0115 end
0116
0117 %-----
0118 % Get Default Plist
0119 %-----
0120 function pl_default = getDefaultPlist()
0121     pl_default = plist('vertices', [], 'method', 'spline');
0122 end
0123

```

ltpda_spikecleaning

PURPOSE ^

LTPDA_spikecleaning detects and corrects possible spikes in analysis objects

SYNOPSIS ^

function varargout=ltpda_spikecleaning(varargin)

DESCRIPTION ^

```
LTPDA_spikecleaning detects and corrects possible spikes in analysis objects

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_ SPIKECLEANING detects spikes in the temperature data and
              replaces them by artificial values depending on the method chosen ('random',
              'mean', 'previous').
              Spikes are defined as singular samples with an (absolute) value
              higher than kspike times the standard deviation of the high-pass
              filtered (IIR filter) input A0.

CALL:        b = ltpda_spikecleaning(a1, a2, ..., an, pl)

INPUTS:      aN - a list of analysis objects
              pl - parameter list

OUTPUTS:     b - a list of analysis objects with "spike values" removed
              and corrected

PARAMETERES: 'kspike' - set the kspike value. High values imply
                    not correction of relative low amplitude spike
                    [default: 3.3]
              'method' - method used to replace the spike value: 'random',
                    'mean', 'previous' [default: random]
              'fc' - frequency cut-off of the IIR filter [default: 0.025]
              'order' - order of the IIR filter [default: 2]
              'ripple' - specify pass/stop-band ripple for bandpass
                    and bandreject filters
                    <<default: 0.5>>

VERSION:     $Id: ltpda_spikecleaning.m,v 1.5 2008/08/10 20:24:50 hewitson Exp $

HISTORY:     24-12-2007 J Sanjuan
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_spikecleaning detects and corrects possible spikes in analysis objects
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_ SPIKECLEANING detects spikes in the temperature data and
0006 %              replaces them by artificial values depending on the method chosen
('random',
0007 %              'mean', 'previous').
0008 %              Spikes are defined as singular samples with an (absolute) value
0009 %              higher than kspike times the standard deviation of the high-pass
0010 %              filtered (IIR filter) input A0.
```

```
0011 %
0012 % CALL:      b = ltpda_spikecleaning(a1, a2, ..., an, pl)
0013 %
0014 % INPUTS:    aN - a list of analysis objects
0015 %            pl - parameter list
0016 %
0017 % OUTPUTS:   b - a list of analysis objects with "spike values" removed
0018 %            and corrected
0019 %
0020 % PARAMETERES: 'kspike' - set the kspike value. High values imply
0021 %               not correction of relative low amplitude spike
0022 %               [default: 3.3]
0023 %               'method' - method used to replace the spike value: 'random,
0024 %               'mean', 'previous' [default:random]
0025 %               'fc' - frequency cut-off of the IIR filter [default: 0.025]
0026 %               'order' - order of the IIR filter [default: 2]
0027 %               'ripple' - specify pass/stop-band ripple for bandpass
0028 %               and bandreject filters
0029 %               <<default: 0.5>>
0030 %
0031 % VERSION:    $Id: ltpda_spikecleaning.m,v 1.5 2008/08/10 20:24:50 hewitson Exp $
0032 %
0033 % HISTORY:    24-12-2007 J Sanjuan
0034 %             Creation
0035 %
0036 %%%%%%%%%%%
0037
0038 function varargout=ltpda_spikecleaning(varargin)
0039 warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/spikecleaning now.');
```

```
0040
0041 % Call ao/spikecleaning
0042 bs = spikecleaning(varargin{:});
0043
0044 % set output
0045 varargout{1} = bs;
0046
0047 end
0048
```

ltpda_gapfilling

PURPOSE ^

LTPDA_GAPFILLING fills possible gaps in data.

SYNOPSIS ^

function varargout=ltpda_gapfilling(varargin)

DESCRIPTION ^

```
LTPDA_GAPFILLING fills possible gaps in data.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_GAPFILLING interpolated data between two data
              segments. This function might be useful for possible
              gaps or corrupted data. Two different types of
              interpolating are available: linear and spline, the latter
              results in a smoother curve between the two data segments.

CALL:        b = ltpda_spikecleaning(a1, a2, pl)

INPUTS:      a1 - data segment previous to the gap
              a2 - data segment posterior to the gap
              pl - parameter list

OUTPUTS:     b - data segment containing a1, a2 and the filled data
              segment, i.e., b=[a1 dataare_filled a2].

PARAMETERES: 'method' - method used to interpolate data between a1 and a2.
              Two options can be used: 'linear' and 'spline'.
              Default values is 'linear'.
              'addnoise' - noise can be added to the interpolated data.
              This noise is defined as random variable with
              zero mean and variance equal to the high-frequency
              noise if a1. 'Yes' adds noise. Default value
              is 'no'.

VERSION:     $Id: ltpda_gapfilling.m,v 1.4 2008/08/10 20:24:50 hewitson Exp $

HISTORY:     28-12-2007 J Sanjuan
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

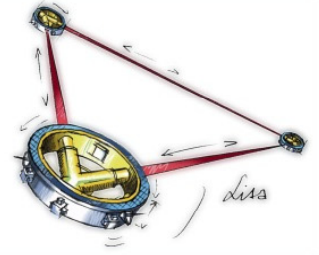
This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_GAPFILLING fills possible gaps in data.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_GAPFILLING interpolated data between two data
0006 %               segments. This function might be useful for possible
0007 %               gaps or corrupted data. Two different types of
0008 %               interpolating are available: linear and spline, the latter
0009 %               results in a smoother curve between the two data segments.
0010 %
0011 % CALL:        b = ltpda_spikecleaning(a1, a2, pl)
0012 %
0013 % INPUTS:      a1 - data segment previous to the gap
```

```
0014 %           a2 - data segment posterior to the gap
0015 %           pl - parameter list
0016 %
0017 % OUTPUTS:      b - data segment containing a1, a2 and the filled data
0018 %                segment, i.e., b=[a1 dataare_filled a2].
0019 %
0020 % PARAMETERES:  'method' - method used to interpolate data between a1 and a2.
0021 %                Two options can be used: 'linear' and 'spline'.
0022 %                Default values is 'linear'.
0023 %                'addnoise' - noise can be added to the interpolated data.
0024 %                This noise is defined as random variable with
0025 %                zero mean and variance equal to the high-frequency
0026 %                noise if a1. 'Yes' adds noise. Default value
0027 %                is 'no'.
0028 %
0029 % VERSION:      $Id: ltpda_gapfilling.m,v 1.4 2008/08/10 20:24:50 hewitson Exp $
0030 %
0031 % HISTORY:      28-12-2007 J Sanjuan
0032 %                Creation
0033 %
0034 %%%%%%%%%%%%%%
0035 %
0036 function varargout=ltpda_gapfilling(varargin)
0037
0038     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/gapfilling now.');
```

```
0039
0040     % Call ao/gapfilling
0041     bs = gapfilling(varargin{:});
0042
0043     % set output
0044     varargout{1} = bs;
0045
0046 end
```



Improved spectrum estimation from digitized time series on a logarithmic frequency axis

[1] M. Tröbs, G. Heinzel. Improved spectrum estimation from digitized time series on a logarithmic frequency axis. *MES*, 39:20–129, oct 2005.

Abstract:

We present a practical technique for spectrum and spectral density estimation from long time series by Fourier transforms. We apply Welch's popular technique of "averaging over modified periodograms" which uses Fourier transforms of fixed length with time-domain windows and overlap. Our technique retains the basic properties of this method, but computes the optimal frequency resolution individually for each Fourier frequency on a logarithmic frequency axis, thus yielding results that are more useful than those of the standard techniques.

ltpda_pwelch

PURPOSE ^

LTPDA_PWELCH makes power spectral density estimates of the time-series objects

SYNOPSIS ^

function varargout = ltpda_pwelch(varargin)

DESCRIPTION ^

```
LTPDA_PWELCH makes power spectral density estimates of the time-series objects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_PWELCH makes power spectral density estimates of the
              time-series objects in the input analysis objects.

              This function is deprecated and will be removed in future versions.
              This just calls ao/pwelch now.

VERSION:      $Id: ltpda_pwelch.m,v 1.22 2008/08/10 20:16:49 hewitson Exp $

HISTORY:      07-02-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

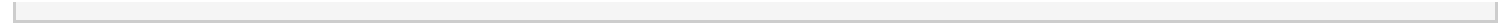
This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_PWELCH makes power spectral density estimates of the time-series objects
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_PWELCH makes power spectral density estimates of the
0006 %              time-series objects in the input analysis objects.
0007 %
0008 %              This function is deprecated and will be removed in future versions.
0009 %              This just calls ao/pwelch now.
0010 %
0011 % VERSION:      $Id: ltpda_pwelch.m,v 1.22 2008/08/10 20:16:49 hewitson Exp $
0012 %
0013 % HISTORY:      07-02-2007 M Hewitson
0014 %              Creation
0015 %
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 function varargout = ltpda_pwelch(varargin)
0020
0021     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/psd now.');
```

```
0022
0023     % Call ao/psd
0024     bs = psd(varargin{:});
0025
0026     % set output
0027     varargout{1} = bs;
0028
0029 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

specwin

PURPOSE ^

% SPECWIN spectral window object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
% SPECWIN spectral window object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SPECWIN spectral window object class constructor.
             Create a spectral window from libSpecWin.

SUPER CLASSES: ltpda_nuo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
        version - cvs-version string.

    Protected Properties (read only)
        type      - name of window object
        alpha     - alpha parameter for various window functions
        psll      - peak sidelobe level
        rov       - recommended overlap
        nenbw     - normalised equivalent noise bandwidth
        w3db      - 3 dB bandwidth in bins
        flatness  - window flatness
        ws        - sum of window values
        ws2       - sum of squares of window values
        win       - window samples (column vector)

CONSTRUCTOR:

    w = specwin()
        creates an empty object

    w = specwin(w)
        copies a specwin object

    w = specwin('name', N)
        creates a specwin object of a particular type and length.
        (see below for supported types.)

    w = specwin('Kaiser', N, psll)
        create a specwin Kaiser window with the prescribed psll.

'name' should be one of the following standard windows:

    Rectangular, Welch, Bartlett, Hanning, Hamming,
    Nuttall3, Nuttall4, Nuttall3a, Nuttall3b, Nuttall4a
    Nuttall4b, Nuttall4c, BH92, SFT3F, SFT3M, FTNI, SFT4F, SFT5F
    SFT4M, FTHP, HFT70, FTSRS, SFT5M, HFT90D, HFT95, HFT116D
    HFT144D, HFT169D, HFT196D, HFT223D, HFT248D

M-FILE INFO: The following call returns an minfo object that contains
              information about the specwin constructor:
                >> info = specwin.getInfo
              or  >> info = specwin.getInfo('specwin')

              You can get information about class methods by calling:
                >> info = specwin.getInfo(method)
              e.g. >> info = specwin.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
                >> info = specwin.getInfo(method, set)
              e.g. >> info = specwin.getInfo('specwin', 'Default')
```

VERSION: \$Id: specwin.m,v 1.53 2008/09/03 16:39:05 hewitson Exp \$

HISTORY: 30-01-07 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of specwin objects depending of the second input
-  [get_window](#) GET_WINDOW returns the required window function as a structure.
-  [kaiser_alpha](#) KAISER_ALPHA returns the alpha parameter that gives the required input
-  [kaiser_flatness](#) KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser
-  [kaiser_nenbw](#) KAISER_NENBW returns the normalized noise-equivalent bandwidth for a
-  [kaiser_rov](#) KAISER_ROV returns the recommended overlap for a Kaiser window with
-  [kaiser_w3db](#) KAISER_W3DB returns the 3dB bandwidth in bins of a kaiser window with
-  [specwin](#) % SPECWIN spectral window object class constructor.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version
-  [win_BH92](#) WIN_BH92 return BH92 window sample for input z.
-  [win_Bartlett](#) WIN_BARTLETT return Bartlett window sample for input z.
-  [win_FTHP](#) WIN_FTHP return FTHP window sample for input z.
-  [win_FTNl](#) WIN_FTNl return FTNI window sample for input z.
-  [win_FTSRS](#) WIN_FTSRS return FTSRS window sample for input z.
-  [win_HFT116D](#) WIN_HFT116D return HFT116D window sample for input z.
-  [win_HFT144D](#) WIN_HFT144D return HFT144D window sample for input z.
-  [win_HFT169D](#) WIN_HFT169D return HFT169D window sample for input z.
-  [win_HFT196D](#) WIN_HFT196D return HFT196D window sample for input z.
-  [win_HFT223D](#) WIN_HFT223D return HFT223D window sample for input z.
-  [win_HFT248D](#) WIN_HFT248D return HFT248D window sample for input z.
-  [win_HFT70](#) WIN_HFT70 return HFT70 window sample for input z.
-  [win_HFT90D](#) WIN_HFT90D return HFT90D window sample for input z.
-  [win_HFT95](#) WIN_HFT95 return HFT95 window sample for input z.
-  [win_Hamming](#) WIN_HAMMING return Hamming window sample for input z.
-  [win_Hanning](#) WIN_HANNING return Hanning window sample for input z.
-  [win_Nuttall3](#) WIN_NUTTALL3 return Nuttall3 window sample for input z.
-  [win_Nuttall3a](#) WIN_NUTTALL3A return Nuttall3a window sample for input z.
-  [win_Nuttall3b](#) WIN_NUTTALL3B return Nuttall3b window sample for input z.
-  [win_Nuttall4](#) WIN_NUTTALL4 return Nuttall4 window sample for input z.
-  [win_Nuttall4a](#) WIN_NUTTALL4A return Nuttall4a window sample for input z.
-  [win_Nuttall4b](#) WIN_NUTTALL4B return Nuttall4b window sample for input z.
-  [win_Nuttall4c](#) WIN_NUTTALL4C return Nuttall4c window sample for input z.
-  [win_Rectangular](#) WIN_RECTANGULAR return rectangular window sample for input z.
-  [win_SFT3F](#) WIN_SFT3F return SFT3F window sample for input z.
-  [win_SFT3M](#) WIN_SFT3M return SFT3M window sample for input z.
-  [win_SFT4F](#) WIN_SFT4F return SFT4F window sample for input z.
-  [win_SFT4M](#) WIN_SFT4M return SFT4M window sample for input z.
-  [win_SFT5F](#) WIN_SFT5F return SFT5F window sample for input z.
-  [win_SFT5M](#) WIN_SFT5M return SFT5M window sample for input z.
-  [win_Welch](#) WIN_WELCH return Welch window sample for input z.

This function is called by:

-  [copy](#) COPY Make copy of specwin objects depending of the second input

 **specwin** % SPECWIN spectral window object class constructor.

SUBFUNCTIONS

 **function** ww = specwin(varargin)
 **function** ww = getTypes()
 **function** out = VEROUT()
 **function** ii = getInfo(varargin)
 **function** out = SETS()
 **function** out = getDefaultPlist(set)

SOURCE CODE

```
0001 % % SPECWIN spectral window object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SPECWIN spectral window object class constructor.
0005 %             Create a spectral window from libSpecWin.
0006 %
0007 % SUPER CLASSES: ltpda_nuo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         version - cvs-version string.
0013 %
0014 %     Protected Properties (read only)
0015 %         type - name of window object
0016 %         alpha - alpha parameter for various window functions
0017 %         psll - peak sidelobe level
0018 %         rov - recommended overlap
0019 %         nenbw - normalised equivalent noise bandwidth
0020 %         w3db - 3 dB bandwidth in bins
0021 %         flatness - window flatness
0022 %         ws - sum of window values
0023 %         ws2 - sum of squares of window values
0024 %         win - window samples (column vector)
0025 %
0026 % CONSTRUCTOR:
0027 %
0028 %     w = specwin()
0029 %         creates an empty object
0030 %
0031 %     w = specwin(w)
0032 %         copies a specwin object
0033 %
0034 %     w = specwin('name', N)
0035 %         creates a specwin object of a particular type and length.
0036 %         (see below for supported types.)
0037 %
0038 %     w = specwin('Kaiser', N, psll)
0039 %         create a specwin Kaiser window with the prescribed psll.
0040 %
0041 % 'name' should be one of the following standard windows:
0042 %
0043 %     Rectangular, Welch, Bartlett, Hanning, Hamming,
0044 %     Nuttall3, Nuttall4, Nuttall3a, Nuttall3b, Nuttall4a
0045 %     Nuttall4b, Nuttall4c, BH92, SFT3F, SFT3M, FTNI, SFT4F, SFT5F
0046 %     SFT4M, FTHP, HFT70, FTSRS, SFT5M, HFT90D, HFT95, HFT116D
0047 %     HFT144D, HFT169D, HFT196D, HFT223D, HFT248D
0048 %
0049 % M-FILE INFO: The following call returns an minfo object that contains
0050 %               information about the specwin constructor:
0051 %               >> info = specwin.getInfo
0052 %               or   >> info = specwin.getInfo('specwin')
0053 %
0054 %               You can get information about class methods by calling:
0055 %               >> info = specwin.getInfo(method)
0056 %               e.g. >> info = specwin.getInfo('eq')
0057 %
0058 %               You can also restrict the sets of parameters contained in
0059 %               the minfo object by calling:
0060 %               >> info = specwin.getInfo(method, set)
0061 %               e.g. >> info = specwin.getInfo('specwin', 'Default')
0062 %
0063 % VERSION:      $Id: specwin.m,v 1.53 2008/09/03 16:39:05 hewitson Exp $
0064 %
```

```

0065 % HISTORY:      30-01-07 M Hewitson
0066 %              Creation
0067 %
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 %              w = specwin('Flattop', N, psll) - create a specwin Flattop window
0071 %              with the prescribed psll.
0072 %
0073 classdef specwin < ltpda_nuo
0074 %
0075 %----- Private read-only Properties -----
0076 %----- Private read-only Properties -----
0077 %----- Private read-only Properties -----
0078 properties (SetAccess = private)
0079     type      = ' ';
0080     alpha     = [ ]';
0081     psll      = [ ]';
0082     rov       = [ ]';
0083     nenbw     = [ ]';
0084     w3db      = [ ]';
0085     flatness  = [ ]';
0086     ws        = [ ]';
0087     ws2       = [ ]';
0088     win       = [ ]';
0089 end
0090
0091 properties (SetAccess = protected)
0092     version = '$Id: specwin.m,v 1.53 2008/09/03 16:39:05 hewitson Exp $';
0093 end
0094 %
0095 %----- Declaration of public methods -----
0096 %----- Declaration of public methods -----
0097 %----- Declaration of public methods -----
0098 methods
0099 %
0100 %----- Property rules -----
0101 %----- Property rules -----
0102 %----- Property rules -----
0103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104 %              Constructor
0105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0107 function ww = specwin(varargin)
0108 %
0109     import utils.const.*
0110     utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0111 %
0112     %% Call superclass
0113     ww = ww@ltpda_nuo(varargin{:});
0114 %
0115     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set default values %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116 %
0117     switch nargin
0118     case 0
0119         utils.helper.msg(msg.OPROC1, 'empty constructor');
0120     case 1
0121         if isa(varargin{1}, 'specwin')
0122             utils.helper.msg(msg.OPROC1, 'copy constructor');
0123             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% spw = specwin(specwin-object) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0124             % copy existing specwin
0125             ww = copy(varargin{1}, 1);
0126         elseif isstruct(varargin{1})
0127             utils.helper.msg(msg.OPROC1, 'constructing from struct');
0128             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% spw = specwin(struct) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0129             %
0130             %% Set properties which are declared in this class
0131             spw_struct = varargin{1};
0132 %
0133             ww.type      = spw_struct.type;
0134             ww.alpha     = spw_struct.alpha;
0135             ww.psll      = spw_struct.psll;
0136             ww.rov       = spw_struct.rov;
0137             ww.nenbw     = spw_struct.nenbw;
0138             ww.w3db      = spw_struct.w3db;
0139             ww.flatness  = spw_struct.flatness;
0140             ww.ws        = spw_struct.ws;
0141             ww.ws2       = spw_struct.ws2;
0142             ww.win       = spw_struct.win;
0143             ww.version   = spw_struct.version;
0144 %
0145             elseif ischar(varargin{1})
0146                 utils.helper.msg(msg.OPROC1, 'constructing from string');
0147                 error('### Code up the char constructor.');
```

```

0151         if ischar(varargin{1})
0152             %%%%%%%%%%% spw = specwin('Win_type', N) %%%%%%%%%%%
0153             ww = get_window(ww, varargin{1}, varargin{2});
0154         else
0155             error('### Unknown 2 argument constructor for specwin object.')
0156         end
0157         case 3
0158             utils.helper.msg(msg.OPROC1, 'constructing type %s', varargin{1});
0159             %%%%%%%%%%% spw = specwin('Kaiser', N, psll) %%%%%%%%%%%
0160             ww = get_window(ww, varargin{1}, varargin{2}, varargin{3});
0161         otherwise
0162             error('### Unknown number of constructor arguments');
0163         end % End of constructor
0164     end
0165 end
0166
0167     varargout = copy(varargin)
0168
0169 end % End public methods
0170
0171 %%%%%%%%%%%
0172 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0173 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Methods (Static, Public) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0174 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0175
0176 methods (Static = true)
0177
0178     varargout = update_struct(varargin);
0179
0180     function ww = getTypes()
0181         ww = {
0182             'Rectangular', 'Welch', 'Bartlett', 'Hanning', 'Hamming', ...
0183             'Nuttall3', 'Nuttall4', 'Nuttall3a', 'Nuttall3b', 'Nuttall4a', ...
0184             'Nuttall4b', 'Nuttall4c', 'BH92', 'SFT3F', 'SFT3M', 'FTNI', 'SFT4F', 'SFT5F', ...
0185             'SFT4M', 'FTHP', 'HFT70', 'FTSRS', 'SFT5M', 'HFT90D', 'HFT95', 'HFT116D', ...
0186             'HFT144D', 'HFT169D', 'HFT196D', 'HFT223D', 'HFT248D' ...
0187         };
0188     end
0189
0190     function out = VEROUT()
0191         out = '$Id: specwin.m,v 1.53 2008/09/03 16:39:05 hewitson Exp $';
0192     end
0193
0194     function ii = getInfo(varargin)
0195         ii = utils.helper.generic_getInfo(varargin{:}, 'specwin');
0196     end
0197
0198     function out = SETS()
0199         out = {'Default'};
0200     end
0201
0202     function out = getDefaultPlist(set)
0203         switch set
0204             case 'Default'
0205                 out = plist();
0206             otherwise
0207                 out = plist();
0208         end
0209     end
0210
0211 end % End static methods
0212
0213 %%%%%%%%%%%
0214 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0215 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Methods (Static, Private) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0216 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0217
0218 methods (Static = true, Access = private)
0219
0220     varargout = kaiser_alpha(varargin)
0221     varargout = kaiser_flatness(varargin)
0222     varargout = kaiser_nenbw(varargin)
0223     varargout = kaiser_rov(varargin)
0224     varargout = kaiser_w3db(varargin)
0225
0226 end % End static private methods
0227
0228 %%%%%%%%%%%
0229 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0230 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Methods (protected) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0231 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0232
0233 methods (Access = protected)
0234     varargout = get_window(varargin)
0235     varargout = win_BH92(varargin)
0236     varargout = win_Bartlett(varargin)
0237     varargout = win_FTHP(varargin)
0238     varargout = win_FTNI(varargin)

```

```
0237     varargout = win_FTSRS(varargin)
0238     varargout = win_HFT116D(varargin)
0239     varargout = win_HFT144D(varargin)
0240     varargout = win_HFT169D(varargin)
0241     varargout = win_HFT196D(varargin)
0242     varargout = win_HFT223D(varargin)
0243     varargout = win_HFT248D(varargin)
0244     varargout = win_HFT70(varargin)
0245     varargout = win_HFT90D(varargin)
0246     varargout = win_HFT95(varargin)
0247     varargout = win_Hamming(varargin)
0248     varargout = win_Hanning(varargin)
0249     varargout = win_Nuttall3(varargin)
0250     varargout = win_Nuttall3a(varargin)
0251     varargout = win_Nuttall3b(varargin)
0252     varargout = win_Nuttall4(varargin)
0253     varargout = win_Nuttall4a(varargin)
0254     varargout = win_Nuttall4b(varargin)
0255     varargout = win_Nuttall4c(varargin)
0256     varargout = win_Rectangular(varargin)
0257     varargout = win_SFT3F(varargin)
0258     varargout = win_SFT3M(varargin)
0259     varargout = win_SFT4F(varargin)
0260     varargout = win_SFT4M(varargin)
0261     varargout = win_SFT5F(varargin)
0262     varargout = win_SFT5M(varargin)
0263     varargout = win_Welch(varargin)
0264 end
0265
0266 end % End classdef
0267
0268
0269
0270
```


ltpda_cpsd

PURPOSE ^

LTPDA_CPSD makes cross-spectral density estimates of the time-series objects.

SYNOPSIS ^

function varargout = ltpda_cpsd(varargin)

DESCRIPTION ^

```
LTPDA_CPSD makes cross-spectral density estimates of the time-series objects.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_CPSD makes cross-spectral density estimates of the
time-series objects in the input analysis objects. CPSDs are computed
using MATLAB's cpsd (>> help cpsd).

CALL:      b = ltpda_cpsd(a1,a2,a3,...,pl)

INPUTS:    b      - output analysis objects
            aN     - input analysis objects (at least two)
            pl     - input parameter list

            The function makes CPSD estimates between all input AOs.
            Therefore, if the input argument list contains N analysis objects, the
            output a will contain NxN CPSD estimates.
            The diagonal elements will be S_ai_ai and will be equivalent to
            the output of ltpda_pwelch(ai)

            If the last input argument is a parameter list (plist) it is used. The
            following parameters are recognised.

PARAMETERS: 'Win'      - a specwin window object [default: Kaiser -200dB psll]
            'Olap'     - segment percent overlap [default: taken from window function]
            'Nfft'     - number of samples in each fft [default: length of input data]
            'Order'    - order of segment detrending:
                        -1 - no detrending
                        0 - subtract mean [default]
                        1 - subtract linear fit
                        N - subtract fit of polynomial, order N

VERSION:    $Id: ltpda_cpsd.m,v 1.12 2008/08/10 20:16:49 hewitson Exp $

HISTORY:    07-02-2007 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_CPSD makes cross-spectral density estimates of the time-series objects.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_CPSD makes cross-spectral density estimates of the
0006 %               time-series objects in the input analysis objects. CPSDs are computed
0007 %               using MATLAB's cpsd (>> help cpsd).
0008 %
0009 % CALL:      b = ltpda_cpsd(a1,a2,a3,...,pl)
```



```
0010 %
0011 % INPUTS:      b      - output analysis objects
0012 %              aN     - input analysis objects (at least two)
0013 %              pl     - input parameter list
0014 %
0015 %              The function makes CPSD estimates between all input AOs.
0016 %              Therefore, if the input argument list contains N analysis objects, the
0017 %              output a will contain NxN CPSD estimates.
0018 %              The diagonal elements will be S_ai_ai and will be equivalent to
0019 %              the output of ltpda_pwelch(ai)
0020 %
0021 %              If the last input argument is a parameter list (plist) it is used. The
0022 %              following parameters are recognised.
0023 %
0024 % PARAMETERS: 'Win'   - a specwin window object [default: Kaiser -200dB psll]
0025 %              'Olap' - segment percent overlap [default: taken from window function]
0026 %              'Nfft' - number of samples in each fft [default: length of input data]
0027 %              'Order' - order of segment detrending:
0028 %                      -1 - no detrending
0029 %                      0 - subtract mean [default]
0030 %                      1 - subtract linear fit
0031 %                      N - subtract fit of polynomial, order N
0032 %
0033 % VERSION:      $Id: ltpda_cpsd.m,v 1.12 2008/08/10 20:16:49 hewitson Exp $
0034 %
0035 % HISTORY:      07-02-2007 M Hewitson
0036 %                Creation
0037 %
0038 %%%%%%%%%%%%%%
0039
0040 function varargout = ltpda_cpsd(varargin)
0041
0042     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/cpsd now.');
```

```
0043
0044     % Call ao/cpsd
0045     bs = cpsd(varargin{:});
0046
0047     % set output
0048     varargout{1} = bs;
0049 end
```

ltpda_cohere

PURPOSE ^

LTPDA_COHERE makes coherence estimates of the time-series objects

SYNOPSIS ^

function varargout = ltpda_cohere(varargin)

DESCRIPTION ^

```
LTPDA_COHERE makes coherence estimates of the time-series objects

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_COHERE makes coherence estimates of the time-series objects
              in the input analysis objects. Coherences are computed using
              MATLAB's mscohere (>> help mscohere).

CALL:        b = ltpda_cohere(a1,a2,a3,...,pl)

INPUTS:      b      - output analysis objects
              aN     - input analysis objects (at least two)
              pl     - input parameter list

              The function makes coherence estimates between all
              input AOs. Therefore, if the input argument list contains N
              analysis objects, the output, b, will contain NXN coherence estimates.

              If the last input argument is a parameter list (plist) it is used.
              The following parameters are recognised.

PARAMETERS:  Win      - a specwin window object [default: Kaiser -200dB psll]
              Olap    - segment percent overlap [default: taken from window function]
              Nfft    - number of samples in each fft [default: half length of input data]
              Order   - order of detrending.
                      -1 - no detrending
                      0  - subtract mean [default]
                      1  - subtract linear fit
                      N  - subtract fit of polynomial, order N

VERSION:     $Id: ltpda_cohere.m,v 1.14 2008/08/10 20:16:49 hewitson Exp $

HISTORY:     07-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_COHERE makes coherence estimates of the time-series objects
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_COHERE makes coherence estimates of the time-series objects
0006 %              in the input analysis objects. Coherences are computed using
0007 %              MATLAB's mscohere (>> help mscohere).
0008 %
0009 % CALL:        b = ltpda_cohere(a1,a2,a3,...,pl)
0010 %
```

```

0011 % INPUTS:      b      - output analysis objects
0012 %              aN     - input analysis objects (at least two)
0013 %              pl     - input parameter list
0014 %
0015 %              The function makes coherence estimates between all
0016 %              input AOs. Therefore, if the input argument list contains N
0017 %              analysis objects, the output, b, will contain NXN coherence estimates.
0018 %
0019 %              If the last input argument is a parameter list (plist) it is used.
0020 %              The following parameters are recognised.
0021 %
0022 % PARAMETERS:  Win      - a specwin window object [default: Kaiser -200dB psll]
0023 %              Olap     - segment percent overlap [default: taken from window function]
0024 %              Nfft     - number of samples in each fft [default: half length of input
data]
0025 %              Order    - order of detrending.
0026 %                      -1 - no detrending
0027 %                      0 - subtract mean [default]
0028 %                      1 - subtract linear fit
0029 %                      N - subtract fit of polynomial, order N
0030 %
0031 %
0032 % VERSION:      $Id: ltpda_cohere.m,v 1.14 2008/08/10 20:16:49 hewitson Exp $
0033 %
0034 % HISTORY:      07-02-2007 M Hewitson
0035 %              Creation
0036 %
0037 %%%%%%%%%%%
0038 %
0039 function varargout = ltpda_cohere(varargin)
0040     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/cohere now.');
```

ltpda_tfe

PURPOSE ^

LTPDA_TFE makes transfer function estimates of the time-series objects.

SYNOPSIS ^

function varargout = ltpda_tfe(varargin)

DESCRIPTION ^

```
LTPDA_TFE makes transfer function estimates of the time-series objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_TFE makes transfer function estimates of the time-series
              objects in the input analysis objects. Transfer functions are
              computed using MATLAB's tfestimate (>> help tfestimate).

CALL:        b = ltpda_tfe(a1,a2,a3,...,pl)

INPUTS:      b      - output analysis objects
              aN     - input analysis objects (at least two)
              pl     - input parameter list

OUTPUTS:     The output matrix b contains NxN transfer functions estimates (all
              possible pairs of input AOs).

              If the last input argument is a parameter list (plist) it is used.
              The following parameters are recognised.

PARAMETERS:  'Win'    - a specwin window object [default: Kaiser -200dB psll]
              'Olap'  - segment percent overlap [default: taken from window function]
              'Nfft'  - number of samples in each fft [default: length of input data]
              Order   - order of detrending:
                        -1 - no detrending
                        0 - subtract mean [default]
                        1 - subtract linear fit
                        N - subtract fit of polynomial, order N

VERSION:     $Id: ltpda_tfe.m,v 1.18 2008/08/10 20:16:49 hewitson Exp $

HISTORY:     07-02-2007 M Hewitson
              Creation

The following call returns a parameter list object that contains the
default parameter values:

>> pl = ltpda_tfe('Params')

The following call returns a string that contains the routine CVS version:

>> version = ltpda_tfe('Version')

The following call returns a string that contains the routine category:

>> category = ltpda_tfe('Category')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```

0001 % LTPDA_TFE makes transfer function estimates of the time-series objects.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_TFE makes transfer function estimates of the time-series
0006 %               objects in the input analysis objects. Transfer functions are
0007 %               computed using MATLAB's tfestimate (>> help tfestimate).
0008 %
0009 % CALL:         b = ltpda_tfe(a1,a2,a3,...,pl)
0010 %
0011 % INPUTS:       b      - output analysis objects
0012 %               aN     - input analysis objects (at least two)
0013 %               pl     - input parameter list
0014 %
0015 % OUTPUTS:      The output matrix b contains NxN transfer functions estimates (all
0016 %               possible pairs of input AOs).
0017 %
0018 %               If the last input argument is a parameter list (plist) it is used.
0019 %               The following parameters are recognised.
0020 %
0021 % PARAMETERS:   'Win'    - a specwin window object [default: Kaiser -200dB psll]
0022 %               'Olap'   - segment percent overlap [default: taken from window function]
0023 %               'Nfft'   - number of samples in each fft [default: length of input data]
0024 %               Order    - order of detrending:
0025 %                           -1 - no detrending
0026 %                           0 - subtract mean [default]
0027 %                           1 - subtract linear fit
0028 %                           N - subtract fit of polynomial, order N
0029 %
0030 % VERSION:      $Id: ltpda_tfe.m,v 1.18 2008/08/10 20:16:49 hewitson Exp $
0031 %
0032 % HISTORY:      07-02-2007 M Hewitson
0033 %               Creation
0034 %
0035 % The following call returns a parameter list object that contains the
0036 % default parameter values:
0037 %
0038 % >> pl = ltpda_tfe('Params')
0039 %
0040 % The following call returns a string that contains the routine CVS version:
0041 %
0042 % >> version = ltpda_tfe('Version')
0043 %
0044 % The following call returns a string that contains the routine category:
0045 %
0046 % >> category = ltpda_tfe('Category')
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 function varargout = ltpda_tfe(varargin)
0051
0052     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/tfe now.');
```

ltpda_lpsd

PURPOSE ^

LTPDA_LPSD implement LPSD algorithm for analysis objects.

SYNOPSIS ^

function varargout = ltpda_lpsd(varargin)

DESCRIPTION ^

```
LTPDA_LPSD implement LPSD algorithm for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LTPDA_LPSD implement LPSD algorithm for analysis objects.

>> bs = ltpda_lpsd(as)

Inputs:
  as - array of analysis objects
  pl - parameter list (see below)

Outputs:
  bs - array of analysis objects, one for each input

Parameter list:
  Kdes - desired number of averages (default 100)
  Lmin - minimum segment length (default 0)
  Jdes - number of spectral frequencies to compute (default fs/2)
  win - a specwin window object
        Only the design parameters of the window object are used; the
        window is recomputed for each DFT length inside the lpsd_core
        algorithm.
  Olap - desired overlap percentage (default: taken from window)
  Order - order of detrending
          -1 - no detrending
           0 - subtract mean
           1 - subtract linear fit
           N - subtract fit of polynomial, order N
  Scale - Scaling of output. Choose from:
          AS - Amplitude (linear) Spectrum
          ASD - Amplitude (linear) Spectral Density
          PS - Power Spectrum
          PSD - Power Spectral Density [default]

The following call returns a parameter list object that contains the
default parameter values:

>> pl = ltpda_lpsd('Params')

The following call returns a string that contains the routine CVS version:

>> version = ltpda_lpsd('Version')

The following call returns a string that contains the routine category:

>> category = ltpda_lpsd('Category')

References:
  "Improved spectrum estimation from digitized time series
  on a logarithmic frequency axis", Michael Troebs, Gerhard Heinzl,
  Measurement 39 (2006) 120-129.

M Hewitson 02-02-07

$Id: ltpda_lpsd.m,v 1.21 2008/08/10 20:16:49 hewitson Exp $
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE [^]

```

0001 % LTPDA_LPSD implement LPSD algorithm for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % LTPDA_LPSD implement LPSD algorithm for analysis objects.
0005 %
0006 %   >> bs = ltpda_lpsd(as)
0007 %
0008 %   Inputs:
0009 %       as - array of analysis objects
0010 %       pl - parameter list (see below)
0011 %
0012 %   Outputs:
0013 %       bs - array of analysis objects, one for each input
0014 %
0015 %   Parameter list:
0016 %       Kdes - desired number of averages (default 100)
0017 %       Lmin - minimum segment length (default 0)
0018 %       Jdes - number of spectral frequencies to compute (default fs/2)
0019 %       win - a specwin window object
0020 %           Only the design parameters of the window object are used; the
0021 %           window is recomputed for each DFT length inside the lpsd_core
0022 %           algorithm.
0023 %       Olap - desired overlap percentage (default: taken from window)
0024 %       Order - order of detrending
0025 %           -1 - no detrending
0026 %           0 - subtract mean
0027 %           1 - subtract linear fit
0028 %           N - subtract fit of polynomial, order N
0029 %       Scale - Scaling of output. Choose from:
0030 %           AS - Amplitude (linear) Spectrum
0031 %           ASD - Amplitude (linear) Spectral Density
0032 %           PS - Power Spectrum
0033 %           PSD - Power Spectral Density [default]
0034 %
0035 % The following call returns a parameter list object that contains the
0036 % default parameter values:
0037 %
0038 % >> pl = ltpda_lpsd('Params')
0039 %
0040 % The following call returns a string that contains the routine CVS version:
0041 %
0042 % >> version = ltpda_lpsd('Version')
0043 %
0044 % The following call returns a string that contains the routine category:
0045 %
0046 % >> category = ltpda_lpsd('Category')
0047 %
0048 % References:
0049 %     "Improved spectrum estimation from digitized time series
0050 %     on a logarithmic frequency axis", Michael Troebs, Gerhard Heinzel,
0051 %     Measurement 39 (2006) 120-129.
0052 %
0053 % M Hewitson 02-02-07
0054 %
0055 % $Id: ltpda_lpsd.m,v 1.21 2008/08/10 20:16:49 hewitson Exp $
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function varargout = ltpda_lpsd(varargin)
0060 %
0061 % warning('!!! This function is deprecated and will be removed from future versions of
0062 % LTPDA. Use ao/lpsd now. ');
0063 %
0064 % Call ao/pwelch
0065 bs = lpsd(varargin{:});
0066 %
0067 % set output
0068 varargout{1} = bs;
0069 end
0070 % END

```

ltpda_lcpsd

PURPOSE ^

LTPDA_LCPSTD implement weighted cross-power estimation computed on a log frequency axis.

SYNOPSIS ^

function varargout = ltpda_lcpsd(varargin)

DESCRIPTION ^

```
LTPDA_LCPSTD implement weighted cross-power estimation computed on a log frequency axis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LTPDA_LCPSTD implement weighted cross-power estimation computed on a
log frequency axis.

Usage:
>> txy = ltpda_lcpsd(as)
>> txy = ltpda_lcpsd(as, pl)

Inputs:
as - array of analysis objects
pl - parameter list (see below)

Outputs:
bs - array of analysis objects, one for each input

Parameter list:
Kdes - desired number of averages [default: 100]
Lmin - minimum segment length [default: 0]
Jdes - number of spectral frequencies to compute [default: fs/4]
Win - a specwin window object
      Only the design parameters of the window object are used; the
      window is recomputed for each DFT length inside the lpsd_core
      algorithm. [default: Kaiser with -200dB PSL]
Nolap - desired overlap [default: taken from window]
Order - order of detrending.
        -1 - no detrending
         0 - subtract mean
         1 - subtract linear fit
         N - subtract fit of polynomial, order N

The following call returns a parameter list object that contains the
default parameter values:

>> pl = ltpda_lcpsd('Params')

The following call returns a string that contains the routine CVS version:

>> version = ltpda_lcpsd(ao,'Version')

The following call returns a string that contains the routine category:

>> category = ltpda_lcpsd(ao,'Category')

M Hewitson 02-02-07

$Id: ltpda_lcpsd.m,v 1.14 2008/08/10 20:16:49 hewitson Exp $
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_LCPSTD implement weighted cross-power estimation computed on a log frequency
axis.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % LTPDA_LCPSTD implement weighted cross-power estimation computed on a
0005 % log frequency axis.
0006 %
0007 % Usage:
0008 %     >> txy = ltpda_lcpsd(as)
0009 %     >> txy = ltpda_lcpsd(as, pl)
0010 %
0011 % Inputs:
0012 %     as - array of analysis objects
0013 %     pl - parameter list (see below)
0014 %
0015 % Outputs:
0016 %     bs - array of analysis objects, one for each input
0017 %
0018 % Parameter list:
0019 %     Kdes - desired number of averages [default: 100]
0020 %     Lmin - minimum segment length [default: 0]
0021 %     Jdes - number of spectral frequencies to compute [default: fs/4]
0022 %     Win - a specwin window object
0023 %           Only the design parameters of the window object are used; the
0024 %           window is recomputed for each DFT length inside the lpsd_core
0025 %           algorithm. [default: Kaiser with -200dB PSL]
0026 %     Noverlap - desired overlap [default: taken from window]
0027 %     Order - order of detrending.
0028 %             -1 - no detrending
0029 %             0 - subtract mean
0030 %             1 - subtract linear fit
0031 %             N - subtract fit of polynomial, order N
0032 %
0033 % The following call returns a parameter list object that contains the
0034 % default parameter values:
0035 %
0036 % >> pl = ltpda_lcpsd('Params')
0037 %
0038 % The following call returns a string that contains the routine CVS version:
0039 %
0040 % >> version = ltpda_lcpsd(ao,'Version')
0041 %
0042 % The following call returns a string that contains the routine category:
0043 %
0044 % >> category = ltpda_lcpsd(ao,'Category')
0045 %
0046 % M Hewitson 02-02-07
0047 %
0048 % $Id: ltpda_lcpsd.m,v 1.14 2008/08/10 20:16:49 hewitson Exp $
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051
0052 function varargout = ltpda_lcpsd(varargin)
0053
0054     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/lcpsd now.');
```

ltpda_lcohere

PURPOSE ^

LTPDA_LCOHERE implement weighted coherence estimation computed on a log frequency axis.

SYNOPSIS ^

function varargout = ltpda_lcohere(varargin)

DESCRIPTION ^

LTPDA_LCOHERE implement weighted coherence estimation computed on a log frequency axis.

LTPDA_LCOHERE implement weighted cross-power estimation computed on a log frequency axis.

Usage:
>> txy = ltpda_lcohere(as)
>> txy = ltpda_lcohere(as, pl)
>> [txy, sig2] = ltpda_lcohere(as, pl) % returns variance estimates

Inputs:
as - array of analysis objects
pl - parameter list (see below)

Outputs:
bs - array of analysis objects, one for each input

Parameter list:
Kdes - desired number of averages [default: 100]
Lmin - minimum segment length [default: 0]
Jdes - number of spectral frequencies to compute [default: 1000]
Win - a specwin window object
Only the design parameters of the window object are used; the window is recomputed for each DFT length inside the lpsd_core algorithm. [default: Kaiser with -200dB PSL]
Nolap - desired overlap [default: taken from window]
Order - order of detrending.
-1 - no detrending
0 - subtract mean
1 - subtract linear fit
N - subtract fit of polynomial, order N

The following call returns a parameter list object that contains the default parameter values:

>> pl = ltpda_lcohere('Params')

M Hewitson 02-02-07

\$Id: ltpda_lcohere.m,v 1.12 2008/08/10 20:16:49 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

0001 % LTPDA_LCOHERE implement weighted coherence estimation computed on a log frequency

```

axis.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % LTPDA_LCOHERE implement weighted cross-power estimation computed on a
0005 % log frequency axis.
0006 %
0007 % Usage:
0008 %     >> txy = ltpda_lcohere(as)
0009 %     >> txy = ltpda_lcohere(as, pl)
0010 %     >> [txy, sig2] = ltpda_lcohere(as, pl) % returns variance estimates
0011 %
0012 % Inputs:
0013 %     as - array of analysis objects
0014 %     pl - parameter list (see below)
0015 %
0016 % Outputs:
0017 %     bs - array of analysis objects, one for each input
0018 %
0019 % Parameter list:
0020 %     Kdes - desired number of averages [default: 100]
0021 %     Lmin - minimum segment length [default: 0]
0022 %     Jdes - number of spectral frequencies to compute [default: 1000]
0023 %     Win - a specwin window object
0024 %           Only the design parameters of the window object are used; the
0025 %           window is recomputed for each DFT length inside the lpsd_core
0026 %           algorithm. [default: Kaiser with -200dB PSL]
0027 %     Nolap - desired overlap [default: taken from window]
0028 %     Order - order of detrending.
0029 %           -1 - no detrending
0030 %           0 - subtract mean
0031 %           1 - subtract linear fit
0032 %           N - subtract fit of polynomial, order N
0033 %
0034 % The following call returns a parameter list object that contains the
0035 % default parameter values:
0036 %
0037 % >> pl = ltpda_lcohere('Params')
0038 %
0039 %
0040 % M Hewitson 02-02-07
0041 %
0042 % $Id: ltpda_lcohere.m,v 1.12 2008/08/10 20:16:49 hewitson Exp $
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045
0046 function varargout = ltpda_lcohere(varargin)
0047
0048     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/lcohere now.');
```

```

0049
0050     % Call ao/lcohere
0051     bs = lcohere(varargin{:});
0052
0053     % set output
0054     varargout{1} = bs;
0055 end
0056 % END

```

ltpda_ltfe

PURPOSE ^

LTPDA_LTFE implement transfer-function estimation computed on a log frequency axis.

SYNOPSIS ^

function varargout = ltpda_ltfe(varargin)

DESCRIPTION ^

```
LTPDA_LTFE implement transfer-function estimation computed on a log frequency axis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LTPDA_LTFE implement transfer-function estimation computed on a
log frequency axis.

Usage:
>> tfs = ltpda_ltfe(as)
>> tfs = ltpda_ltfe(as, pl)

Inputs:
as - array of analysis objects
pl - parameter list (see below)

Outputs:
bs - array of analysis objects, one for each input

Parameter list:
Kdes - desired number of averages [default: 100]
Lmin - minimum segment length [default: 0]
Jdes - number of spectral frequencies to compute [default: fs/4]
Win - a specwin window object
      Only the design parameters of the window object are used; the
      window is recomputed for each DFT length inside the ltpda_dft
      algorithm. [default: Kaiser with -200dB PSL]
Nolap - desired overlap [default: taken from window]
Order - order of detrending.
        -1 - no detrending
         0 - subtract mean
         1 - subtract linear fit
         N - subtract fit of polynomial, order N

The following call returns a parameter list object that contains the
default parameter values:

>> pl = ltpda_ltfe('Params')

The following call returns a string that contains the routine CVS version:

>> version = ltpda_ltfe(ao,'Version')

The following call returns a string that contains the routine category:

>> category = ltpda_ltfe(ao,'Category')

M Hewitson 02-02-07

$Id: ltpda_ltfe.m,v 1.17 2008/08/10 20:16:49 hewitson Exp $
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```

0001 % LTPDA_LTFE implement transfer-function estimation computed on a log frequency axis.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % LTPDA_LTFE implement transfer-function estimation computed on a
0005 % log frequency axis.
0006 %
0007 % Usage:
0008 %     >> tfs = ltpda_ltfe(as)
0009 %     >> tfs = ltpda_ltfe(as, pl)
0010 %
0011 % Inputs:
0012 %     as - array of analysis objects
0013 %     pl - parameter list (see below)
0014 %
0015 % Outputs:
0016 %     bs - array of analysis objects, one for each input
0017 %
0018 % Parameter list:
0019 %     Kdes - desired number of averages [default: 100]
0020 %     Lmin - minimum segment length [default: 0]
0021 %     Jdes - number of spectral frequencies to compute [default: fs/4]
0022 %     Win - a specwin window object
0023 %           Only the design parameters of the window object are used; the
0024 %           window is recomputed for each DFT length inside the ltpda_dft
0025 %           algorithm. [default: Kaiser with -200dB PSLL]
0026 %     Noverlap - desired overlap [default: taken from window]
0027 %     Order - order of detrending.
0028 %             -1 - no detrending
0029 %             0 - subtract mean
0030 %             1 - subtract linear fit
0031 %             N - subtract fit of polynomial, order N
0032 %
0033 % The following call returns a parameter list object that contains the
0034 % default parameter values:
0035 %
0036 % >> pl = ltpda_ltfe('Params')
0037 %
0038 % The following call returns a string that contains the routine CVS version:
0039 %
0040 % >> version = ltpda_ltfe(ao,'Version')
0041 %
0042 % The following call returns a string that contains the routine category:
0043 %
0044 % >> category = ltpda_ltfe(ao,'Category')
0045 %
0046 % M Hewitson 02-02-07
0047 %
0048 % $Id: ltpda_ltfe.m,v 1.17 2008/08/10 20:16:49 hewitson Exp $
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 function varargout = ltpda_ltfe(varargin)
0053 %
0054 warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/ltfe now.');
```

```

0055 %
0056 % Call ao/ltfe
0057 bs = ltfe(varargin{:});
0058 %
0059 % set output
0060 varargout{1} = bs;
0061 end
0062 % END

```

polyfit

PURPOSE ^

POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.

SYNOPSIS ^

function varargout = polyfit(varargin)

DESCRIPTION ^

```
POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: POLYFIT overloads polyfit() function of MATLAB for Analysis
Objects.

CALL:      b = polyfit(a, pl)

PARAMETERS: 'N'      - degree of polynomial to fit
             'coeffs' - (optional) coefficients
                    formed e.g. by [p,s] = polyfit(x,y,N);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('polyfit')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('polyfit', 'None')

VERSION:    $Id: polyfit.m,v 1.18 2008/09/05 11:05:29 ingo Exp $

HISTORY:    05-06-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [polyfit](#) POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [detrend](#) DETREND detrends the input analysis object using a polynomial of degree N.
-  [polyfit](#) POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: POLYFIT overloads polyfit() function of MATLAB for Analysis
0005 %               Objects.
0006 %
0007 % CALL:         b = polyfit(a, pl)
0008 %
0009 % PARAMETERS: 'N'      - degree of polynomial to fit
0010 %             'coeffs' - (optional) coefficients
0011 %                   formed e.g. by [p,s] = polyfit(x,y,N);
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %             >> ao.getInfo('polyfit')
0015 %
0016 %             Get information about a specified set-plist by calling:
0017 %             >> ao.getInfo('polyfit', 'None')
0018 %
0019 % VERSION:      $Id: polyfit.m,v 1.18 2008/09/05 11:05:29 ingo Exp $
0020 %
0021 % HISTORY:      05-06-07 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = polyfit(varargin)
0027
0028 % Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 import utils.const.*
0035 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0036
0037 % Collect input variable names
0038 in_names = cell(size(varargin));
0039 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0040
0041 % Collect all AOs and plists
0042 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0043 pl               = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0044
0045 % Combine plists
0046 pl = combine(pl, getDefaultPlist);
0047
0048 % Decide on a deep copy or a modify
0049 bs = copy(as, nargout);
0050
0051 % Degree of polynomial to fit
0052 N   = find(pl, 'N');
0053 coeffs = find(pl, 'coeffs');
0054
0055 %-----
0056 % Loop over input AOs
0057 for j=1:numel(bs)
0058     if isa(bs(j).data, 'cdata')
0059         warning('!!! Can''t fit to cdata objects. Skipping AO %s', ao_invars{j});
0060         bs(j) = [];
0061     else
0062         % Get x values
0063         x = bs(j).data.getX;
0064         % Fit polynomial
0065         if ~isempty(N)
0066             [p,s] = polyfit(x, bs(j).data.getY, N);
0067         else
0068             p = coeffs;
0069             s = [];
0070         end
0071         % make new values from these coefficients
0072         y = polyval(p, bs(j).data.getX);
0073         % Set data
0074         bs(j).data.setXY(x, y);
0075         % Set output name
0076         bs(j).setName(sprintf('polyfit(%s)', ao_invars{j}), 'internal');
0077         % Set extra processing info
0078         bs(j).procinfo = plist('coeffs', p, 'S', s);
0079         % Add history
0080         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0081     end
0082 end
0083
0084 % Set outputs
0085 if nargout > 0
0086     varargout{1} = bs;

```

```
0087     end
0088 end
0089
0090 %-----
0091 % Get Info Object
0092 %-----
0093 function ii = getInfo(varargin)
0094     if nargin == 1 && strcmpi(varargin{1}, 'None')
0095         sets = {};
0096         pl   = [];
0097     else
0098         sets = {'Default'};
0099         pl   = getDefaultPlist;
0100     end
0101     % Build info object
0102     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: polyfit.m,v 1.18
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0103 end
0104
0105 %-----
0106 % Get Default Plist
0107 %-----
0108 function pl_default = getDefaultPlist()
0109     pl_default = plist('N', 1, 'coeffs', []);
0110 end
0111
0112 % END
0113
```


ltpda_lincom

PURPOSE ^

LTPDA_LINCOM make the linear combination of the input analysis objects.

SYNOPSIS ^

function varargout = ltpda_lincom(varargin)

DESCRIPTION ^

```
LTPDA_LINCOM make the linear combination of the input analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_LINCOM make the linear combination of the
             input analysis objects.
CALL:       b = lincom(a1,a2,a3,...,pl)
             b = lincom(a1,a2,a3,...,aN)
             If no parameter list is specified, one of the analysis objects
             should contain a cdata type with the coefficients inside.
INPUTS:     aN - a list of analysis objects of the same type
             pl - a parameter list
OUTPUTS:    b  - output analysis object
             If the last input argument is a parameter list (plist) it is used.
             The following parameters are recognised.
PARAMETERS: 'coeffs' - vector of coefficients, same length as number of input AOs
VERSION:    $Id: ltpda_lincom.m,v 1.15 2008/08/10 20:27:49 hewitson Exp $
HISTORY:    14-02-07 M Hewitson
             Creation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_LINCOM make the linear combination of the input analysis objects.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_LINCOM make the linear combination of the
0006 %             input analysis objects.
0007 %
0008 % CALL:       b = lincom(a1,a2,a3,...,pl)
0009 %             b = lincom(a1,a2,a3,...,aN)
0010 %
0011 %             If no parameter list is specified, one of the analysis objects
0012 %             should contain a cdata type with the coefficients inside.
0013 %
0014 % INPUTS:     aN - a list of analysis objects of the same type
0015 %             pl - a parameter list
0016 %
0017 % OUTPUTS:    b  - output analysis object
0018 %
```

```
0019 %           If the last input argument is a parameter list (plist) it is used.
0020 %           The following parameters are recognised.
0021 %
0022 % PARAMETERS:  'coeffs' - vector of coefficients, same length as number of input AOs
0023 %
0024 % VERSION:     $Id: ltpda_lincom.m,v 1.15 2008/08/10 20:27:49 hewitson Exp $
0025 %
0026 % HISTORY:     14-02-07 M Hewitson
0027 %               Creation.
0028 %
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 function varargout = ltpda_lincom(varargin)
0032     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/lincom now.');
```

minus

PURPOSE ^

MINUS implements minus operator for analysis objects.

SYNOPSIS ^

function varargout = minus(varargin)

DESCRIPTION ^

```
MINUS implements minus operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MINUS implements minus operator for two analysis objects.

CALL:      a = a1-scalar
           a = a1-a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('minus')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('minus', 'None')

VERSION:    $Id: minus.m,v 1.24 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MINUS implements minus operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MINUS implements minus operator for two analysis objects.
0005 %
0006 % CALL:      a = a1-scalar
0007 %           a = a1-a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('minus')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('minus', 'None')
0014 %
```

```

0015 % VERSION:      $Id: minus.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:       01-02-07 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%
0021
0022 function varargout = minus(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargsout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'minus', '-', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pls = [];
0061     else
0062         sets = {'Default'};
0063         pls = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: minus.m,v 1.24
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0067 end
0068
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075
0076

```

mpower

PURPOSE ^

MPOWER implements mpower operator for analysis objects.

SYNOPSIS ^

function varargout = mpower(varargin)

DESCRIPTION ^

```
MPOWER implements mpower operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MPOWER implements mpower operator for analysis objects.

CALL:      a = a1^scalar
           a = a1^a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('mpower')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mpower', 'None')

VERSION:    $Id: mpower.m,v 1.24 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MPOWER implements mpower operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MPOWER implements mpower operator for analysis objects.
0005 %
0006 % CALL:      a = a1^scalar
0007 %           a = a1^a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('mpower')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('mpower', 'None')
0014 %
```

```

0015 % VERSION:      $Id: mpower.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:       01-02-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%
0021
0022 function varargout = mpower(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargsout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'mpower', '^', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: mpower.m,v 1.24
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0066 end
0067
0068 %-----
0069 % Get Default Plist
0070 %-----
0071 function pl_default = getDefaultPlist()
0072     pl_default = plist();
0073 end
0074 % END
0075

```

mrdivide

PURPOSE ^

MRDIVIDE implements mrdivide operator for analysis objects.

SYNOPSIS ^

function varargout = mrdivide(varargin)

DESCRIPTION ^

```
MRDIVIDE implements mrdivide operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MRDIVIDE implements mrdivide operator for analysis objects.

CALL:      a = a1/scalar
           a = a1/a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('mrdivide')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mrdivide', 'None')

VERSION:    $Id: mrdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MRDIVIDE implements mrdivide operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MRDIVIDE implements mrdivide operator for analysis objects.
0005 %
0006 % CALL:      a = a1/scalar
0007 %           a = a1/a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('mrdivide')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('mrdivide', 'None')
0014 %
```

```

0015 % VERSION:      $Id: mrdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:       01-02-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%
0021
0022 function varargout = mrdivide(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargsout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'mrdivide', '/', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pls = [];
0061     else
0062         sets = {'Default'};
0063         pls = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: mrdivide.m,v 1.24
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0067 end
0068
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075 % END
0076

```


mtimes

PURPOSE ^

MTIMES implements mtimes operator for analysis objects.

SYNOPSIS ^

function varargout = mtimes(varargin)

DESCRIPTION ^

```
MTIMES implements mtimes operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MTIMES implements mtimes operator for analysis objects.

CALL:      a = a1*scalar
           a = a1*a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('mtimes')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mtimes', 'None')

VERSION:    $Id: mtimes.m,v 1.28 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MTIMES implements mtimes operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MTIMES implements mtimes operator for analysis objects.
0005 %
0006 % CALL:      a = a1*scalar
0007 %           a = a1*a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('mtimes')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('mtimes', 'None')
0014 %
```

```

0015 % VERSION:      $Id: mtimes.m,v 1.28 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:       01-02-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = mtimes(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargsout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'mtimes', '*', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pls = [];
0061     else
0062         sets = {'Default'};
0063         pls = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: mtimes.m,v 1.28
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0067 end
0068
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075 % END
0076

```

plus

PURPOSE ^

PLUS implements addition operator for analysis objects.

SYNOPSIS ^

function varargout = plus(varargin)

DESCRIPTION ^

```
PLUS implements addition operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PLUS implements addition operator for two analysis objects.

CALL:      a = a1+scalar
           a = a1+a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('plus')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('plus', 'None')

VERSION:    $Id: plus.m,v 1.37 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % PLUS implements addition operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PLUS implements addition operator for two analysis objects.
0005 %
0006 % CALL:      a = a1+scalar
0007 %           a = a1+a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('plus')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('plus', 'None')
0014 %
```

```

0015 % VERSION:      $Id: plus.m,v 1.37 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY: 01-02-07 M Hewitson
0018 %           Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = plus(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'plus', '+', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pls = [];
0061     else
0062         sets = {'Default'};
0063         pls = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: plus.m,v 1.37
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0067 end
0068
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075
0076 % END

```

power

PURPOSE ^

POWER implements power operator for analysis objects.

SYNOPSIS ^

function varargout = power(varargin)

DESCRIPTION ^

```
POWER implements power operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: POWER implements power operator for analysis objects.

CALL:      a = a1.^scalar
           a = a1.^a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('power')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('power', 'None')

VERSION:    $Id: power.m,v 1.22 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % POWER implements power operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: POWER implements power operator for analysis objects.
0005 %
0006 % CALL:      a = a1.^scalar
0007 %           a = a1.^a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('power')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('power', 'None')
0014 %
```

```

0015 % VERSION:      $Id: power.m,v 1.22 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:       01-02-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = power(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargsout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'power', '.*', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: power.m,v 1.22
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0066 end
0067
0068 %-----
0069 % Get Default Plist
0070 %-----
0071 function pl_default = getDefaultPlist()
0072     pl_default = plist();
0073 end
0074
0075 % END
0076

```

rdivide

PURPOSE ^

RDIVIDE implements rdivide operator for analysis objects.

SYNOPSIS ^

function varargout = rdivide(varargin)

DESCRIPTION ^

```
RDIVIDE implements rdivide operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RDIVIDE implements rdivide operator for analysis objects.

CALL:      a = a1./scalar
           a = a1./a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('rdivide')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('rdivide', 'None')

VERSION:    $Id: rdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % RDIVIDE implements rdivide operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RDIVIDE implements rdivide operator for analysis objects.
0005 %
0006 % CALL:      a = a1./scalar
0007 %           a = a1./a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('rdivide')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('rdivide', 'None')
0014 %
```

```

0015 % VERSION:      $Id: rdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:       01-02-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%
0021
0022 function varargout = rdivide(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargsout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'rdivide', './', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pls = [];
0061     else
0062         sets = {'Default'};
0063         pls = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: rdivide.m,v 1.24
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0067 end
0068
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075 % END
0076

```


times

PURPOSE ^

TIMES implements times operator for analysis objects.

SYNOPSIS ^

function varargout = times(varargin)

DESCRIPTION ^

```
TIMES implements times operator for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: TIMES implements times operator for analysis objects.

CALL:      a = a1.*scalar
           a = a1.*a2

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('times')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('times', 'None')

VERSION:    $Id: times.m,v 1.25 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [split](#) SPLIT split an analysis object into the specified segments.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl_default = getDefaultPlist()

SOURCE CODE ^

```
0001 % TIMES implements times operator for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TIMES implements times operator for analysis objects.
0005 %
0006 % CALL:      a = a1.*scalar
0007 %           a = a1.*a2
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('times')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('times', 'None')
0014 %
0015 % VERSION:      $Id: times.m,v 1.25 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      01-02-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%
0021
0022 function varargout = times(varargin)
0023
0024     import utils.const.*
0025
0026     % Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034     % Collect input ao's, plist's and ao variable names
0035     in_names = cell(size(varargin));
0036     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038     % Check for numbers in the other arguments and convert to AOs
0039     args = utils.helper.collect_values(varargin);
0040     [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
0041     pl = utils.helper.collect_objects(args, 'plist');
0042
0043     % Combine with default plist
0044     pl = combine(pl, getDefaultPlist);
0045
0046     % Decide on a deep copy or a modify
0047     bs = copy(as, nargsout);
0048
0049     % Set output
0050     varargout{1} = applyoperator(bs, ao_invars, 'times', '.*', pl, getInfo);
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pls = [];
0061     else
0062         sets = {'Default'};
0063         pls = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: times.m,v 1.25
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0067 end
0068
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075 % END
0076

```

attachm

PURPOSE ^

ATTACHM attach an m file to the analysis object.

SYNOPSIS ^

function varargout = attachm(varargin)

DESCRIPTION ^

```
ATTACHM attach an m file to the analysis object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: ATTACHM attach an m file to the analysis object.

CALL:      a = attachm(a, 'foo.m');
           a = attachm(a, plist);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('attachm')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('attachm', 'None')

VERSION:    $Id: attachm.m,v 1.14 2008/09/05 11:05:28 ingo Exp $

HISTORY:    31-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [min](#) MIN computes the minimum value of the data in the AO.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ATTACHM attach an m file to the analysis object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ATTACHM attach an m file to the analysis object.
0005 %
0006 % CALL:      a = attachm(a, 'foo.m');
0007 %           a = attachm(a, plist);
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('attachm')
0011 %
0012 %              Get information about a specified set-plist by calling:
```

```

0013 %                >> ao.getInfo('attachm', 'None')
0014 %
0015 % VERSION:      $Id: attachm.m,v 1.14 2008/09/05 11:05:28 ingo Exp $
0016 %
0017 % HISTORY:      31-03-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021 %
0022 function varargout = attachm(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 import utils.const.*
0031 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs and plists
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039 [pl, pl_invars,rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0040
0041 % Decide on a deep copy or a modify
0042 bs = copy(as, nargout);
0043
0044 % Combine with default plist
0045 pl = combine(pl, getDefaultPlist);
0046
0047 % Get filename
0048 mfile = find(pl, 'filename');
0049 if isempty(mfile)
0050     % get filename from 'rest'
0051     for jj=1:numel(rest)
0052         if ischar(rest{jj})
0053             mfile = rest{jj};
0054             break;
0055         end
0056     end
0057 end
0058 if isempty(mfile)
0059     error('### No filename found in inputs.');
```

```

0060 end
0061
0062 % Loop over AOs
0063 for jj=1:numel(bs)
0064
0065     % get file parts
0066     [pathstr,name,ext,vers] = fileparts(mfile);
0067     if isempty(ext)
0068         mfile = [mfile '.m'];
0069     end
0070     min = textread(mfile, '%s', 'delimiter', '\n', 'whitespace', '');
0071     mout = '';
0072     for j=1:length(min)
0073         mout = [mout '\n' min{j}];
0074     end
0075
0076     % attach to AO
0077     bs(jj).mfile = mout;
0078     bs(jj).mfilename = strrep(mfile, '','', '');
0079 end
0080
0081 % Set outputs
0082 if nargout > 0
0083     varargout{1} = bs;
0084 end
0085 end
0086
0087 %-----
0088 % Get Info Object
0089 %-----
0090 function ii = getInfo(varargin)
0091 if nargin == 1 && strcmpi(varargin{1}, 'None')
0092     sets = {};
0093     pl = [];
0094 else
0095     sets = {'Default'};
0096     pl = getDefaultPlist;
0097 end
0098 % Build info object

```

```
0099     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: attachm.m,v 1.14
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0100 end
0101 %-----
0102 % Get Default Plist
0103 %-----
0104 function pl_default = getDefaultPlist()
0105     pl_default = plist();
0106 end
0107
0108 % END
0109
0110
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

attachmdl

PURPOSE ^

ATTACHMDL attach an mdl file to the analysis object.

SYNOPSIS ^

function varargout = attachmdl(varargin)

DESCRIPTION ^

```
ATTACHMDL attach an mdl file to the analysis object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: ATTACHMDL attach an mdl file to the analysis object.

CALL:      a = attachmdl(a)

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('attachmdl')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('attachmdl', 'None')

VERSION:    $Id: attachmdl.m,v 1.14 2008/09/05 11:05:28 ingo Exp $

HISTORY:    31-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 **copy** COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  **function ii = getInfo(varargin)**
-  **function pl_default = getDefaultPlist()**

SOURCE CODE ^

```
0001 % ATTACHMDL attach an mdl file to the analysis object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ATTACHMDL attach an mdl file to the analysis object.
0005 %
0006 % CALL:      a = attachmdl(a)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> ao.getInfo('attachmdl')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> ao.getInfo('attachmdl', 'None')
0013 %
0014 % VERSION:    $Id: attachmdl.m,v 1.14 2008/09/05 11:05:28 ingo Exp $
0015 %
0016 % HISTORY:    31-03-07 M Hewitson
0017 %              Creation
```

```

0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = attachmdl(varargin)
0022
0023 % Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 import utils.const.*
0030 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs and plists
0037 as = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038
0039 % Decide on a deep copy or a modify
0040 bs = copy(as, nargout);
0041
0042 % get the current mdl file
0043 mdlfile = bdroot(gcb);
0044
0045 % Check its status
0046 dirty = get_param(bdroot, 'Dirty');
0047 if strcmp(dirty, 'on')
0048     errorDlg('The current Simulink model is unsaved. As such, it can not be attached to
the requested AO. Please save the model and run it again.', 'Unsaved Model');
0049 end
0050
0051 % read it
0052 [pathstr,name,ext] = fileparts(mdlfile);
0053 if isempty(ext)
0054     mdlfile = [mdlfile '.mdl'];
0055 end
0056
0057 mdlfilein = textread(mdlfile, '%s', 'delimiter', '\n', 'whitespace', '');
0058 mdlout = '';
0059 for j=1:length(mdlfilein)
0060     mdlout = [mdlout '\n' mdlfilein{j}];
0061 end
0062
0063 % Now attach it to each input AO
0064 for jj=1:numel(bs)
0065     bs(jj).mdlfile = strrep(mdlout, '\n', '\');
0066     bs(jj).mdlfilename = mdlfile;
0067 end
0068
0069 % Set outputs
0070 if nargout > 0
0071     varargout{1} = bs;
0072 end
0073 end
0074
0075 %-----
0076 % Get Info Object
0077 %-----
0078 function ii = getInfo(varargin)
0079     if nargin == 1 && strcmpi(varargin{1}, 'None')
0080         sets = {};
0081         pl = [];
0082     else
0083         sets = {'Default'};
0084         pl = getDefaultPlist;
0085     end
0086     % Build info object
0087     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: attachmdl.m,v
1.14 2008/09/05 11:05:28 ingo Exp $', sets, pl);
0088 end
0089
0090 %-----
0091 % Get Default Plist
0092 %-----
0093 function pl_default = getDefaultPlist()
0094     pl_default = plist();
0095 end
0096

```

cat

PURPOSE ^

CAT concatenate AOs into a vector.

SYNOPSIS ^

function varargout = cat(varargin)

DESCRIPTION ^

```
CAT concatenate AOs into a vector.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CAT concatenate AOs into a vector.

CALL:      a = cat(a1, a2)

REMARK:    This method is a transparent method, and doesn't add history.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('cat')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('cat', 'None')

VERSION:   $Id: cat.m,v 1.14 2008/09/05 11:05:28 ingo Exp $

HISTORY:   27-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CAT concatenate AOs into a vector.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CAT concatenate AOs into a vector.
0005 %
0006 % CALL:      a = cat(a1, a2)
0007 %
0008 % REMARK:    This method is a transparent method, and doesn't add history.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %              >> ao.getInfo('cat')
0012 %
0013 %              Get information about a specified set-plist by calling:
0014 %              >> ao.getInfo('cat', 'None')
0015 %
```



```

0016 % VERSION:      $Id: cat.m,v 1.14 2008/09/05 11:05:28 ingo Exp $
0017 %
0018 % HISTORY:      27-03-07 M Hewitson
0019 %              Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%
0022
0023 function varargout = cat(varargin)
0024
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038 % Collect all AOs and plists
0039 as = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040
0041 % Decide on a deep copy or a modify
0042 bs = copy(as, nargout);
0043
0044 % Set output
0045 if nargout == 0
0046     error('### cat cannot be used as a modifier. Please give an output variable.');
```

```

0047 else
0048     varargout{1} = bs;
0049 end
0050 end
0051
0052 %-----
0053 % Get Info Object
0054 %-----
0055 function ii = getInfo(varargin)
0056     if nargin == 1 && strcmpi(varargin{1}, 'None')
0057         sets = {};
0058         pl = [];
0059     else
0060         sets = {'Default'};
0061         pl = getDefaultPlist;
0062     end
0063     % Build info object
0064     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: cat.m,v 1.14
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0065 end
0066
0067 %-----
0068 % Get Default Plist
0069 %-----
0070 function pl_default = getDefaultPlist()
0071     pl_default = plist();
0072 end
0073
0074

```

copy

PURPOSE ^

COPY makes a (deep) copy of the input AOs.

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY makes a (deep) copy of the input AOs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY makes a deep copy of the input AOs.

CALL:      b = copy(a, flag)

INPUTS:    a      - input analysis object
            flag - 1: make a deep copy, 0: return copies of handles

OUTPUTS:    b - copy of inputs

This is a transparent function and adds no history.

VERSION:    $Id: copy.m,v 1.14 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    14-07-2008 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [t0](#) T0 Get the data property 't0'.

This function is called by:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [acos](#) ACOS overloads the acos method for Analysis objects.
-  [ao](#) AO analysis object class constructor.
-  [asin](#) ASIN overloads the asin method for Analysis objects.
-  [atan](#) ATAN overloads the atan method for Analysis objects.
-  [atan2](#) ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse tangent.
-  [attachm](#) ATTACHM attach an m file to the analysis object.
-  [attachmdl](#) ATTACHMDL attach an mdl file to the analysis object.
-  [cat](#) CAT concatenate AOs into a vector.
-  [complex](#) COMPLEX overloads the complex operator for Analysis objects.
-  [compute](#) COMPUTE performs the given operations on the input AOs.
-  [conj](#) CONJ overloads the conjugate operator for Analysis objects.
-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [cos](#) COS overloads the cos operator for Analysis objects. Cosine of argument in radians.

-  **ctranspose** CTRANSPOSE overloads the ' operator for Analysis Objects.
-  **delay** DELAY delays a time-series using various methods.
-  **det** DET overloads the determinant function for Analysis objects.
-  **detrend** DETREND detrends the input analysis object using a polynomial of degree N.
-  **dft** DFT computes the DFT of the input time-series at the requested frequencies.
-  **diag** DIAG overloads the diagonal operator for Analysis Objects.
-  **diff** DIFF differentiates the data in AO.
-  **dopplercorr** FOO description for function 'foo' in one line. Necessary for lookfor functionality.
-  **downsample** DOWNSAMPLE AOs containing time-series data.
-  **dropduplicates** DROPDUPLICATES drops all duplicate samples in time-series AOs.
-  **dsmean** DSMEAN performs a simple downsampling by taking the mean of every N samples.
-  **eig** EIG overloads the determinant function for Analysis objects.
-  **exp** EXP overloads the exp operator for Analysis objects. Exponential.
-  **fft** FFT overloads the fft method for Analysis objects.
-  **filter** FILTER overrides the filter function for analysis objects.
-  **filtfilt** FILTFILT overrides the filtfilt function for analysis objects.
-  **find** FIND particular samples that satisfy the input query and return a new AO.
-  **firwhiten** FIRWHITEN whitens the input time-series by building an FIR whitening filter.
-  **fixfs** FIXFS resamples the input time-series to have a fixed sample rate.
-  **hist** HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
-  **ifft** IFFT overloads the ifft operator for Analysis objects.
-  **imag** IMAG overloads the imaginary operator for Analysis objects.
-  **index** INDEX index into an AO array or matrix. This properly captures the history.
-  **interp** INTERP interpolate the values in the input AO(s) at new values.
-  **interpmissing** INTERPMISSING interpolate missing samples in a time-series.
-  **inv** INV overloads the inverse function for Analysis Objects.
-  **lincom** LINCOM
-  **linedetect** LINEDETECT find spectral lines in the ao/fsdata objects.
-  **ln** LN overloads the log operator for Analysis objects. Natural logarithm.
-  **log** LOG overloads the log operator for Analysis objects. Natural logarithm.
-  **log10** LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.
-  **lpsd** LPSD implement the LPSD algorithm for analysis objects.
-  **lxspecc** LXSPEC performs log-scale cross-spectral analysis of various forms.
-  **max** MAX computes the maximum value of the data in the AO.
-  **mdc1_ifo2acc_fd** MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
-  **mdc1_ifo2acc_inloop** MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.
-  **mdc1_ifo2control** MDC1_IFO2CONTROL converts the input time-series to control forces.
-  **mdc1_x2acc** MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter
-  **mean** MEAN computes the mean value of the data in the AO.
-  **median** MEDIAN computes the median value of the data in the AO.
-  **min** MIN computes the minimum value of the data in the AO.
-  **minus** MINUS implements minus operator for analysis objects.
-  **mode** MODE computes the modal value of the data in the AO.
-  **mpower** MPOWER implements mpower operator for analysis objects.
-  **mrdivide** MRDIVIDE implements mrdivide operator for analysis objects.
-  **mtimes** MTIMES implements mtimes operator for analysis objects.
-  **norm** NORM overloads the norm operator for Analysis Objects.
-  **phase** PHASE overloads the ltpda_phase operator for Analysis objects.
-  **plus** PLUS implements addition operator for analysis objects.
-  **polyfit** POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.

-  **power** POWER implements power operator for analysis objects.
-  **psd** PSD makes power spectral density estimates of the time-series objects
-  **rdivide** RDIVIDE implements rdivide operator for analysis objects.
-  **real** REAL overloads the real operator for Analysis objects.
-  **resample** RESAMPLE overloads resample function for AOs.
-  **rms** RMS Calculate RMS deviation from spectrum
-  **select** SELECT select particular samples from the input AOs and return new AOs with only those samples.
-  **setDescription** SETDESCRIPTION sets the 'description' property of the ao.
-  **setFs** SETFS sets the 'fs' property of the ao.
-  **setT0** SETT0 sets the 't0' property of the ao.
-  **setX** SETX sets the 'x' property of the ao.
-  **setXY** SETXY sets the 'xy' property of the ao.
-  **setXunits** SETXUNITS sets the 'xunits' property of the ao.
-  **setY** SETY sets the 'y' property of the ao.
-  **setYunits** SETYUNITS sets the 'yunits' property of the ao.
-  **setZ** SETZ sets the 'z' property of the ao.
-  **sign** SIGN overloads the sign operator for Analysis objects.%
-  **sin** SIN overloads the sin method for Analysis objects.
-  **smoother** SMOOTHER smooths a given series of data points using the specified method.
-  **sort** SORT the values in the AO.
-  **split** SPLIT split an analysis object into the specified segments.
-  **sqrt** SQRT computes the square root of the data in the AO.
-  **std** STD computes the standard deviation of the data in the AO.
-  **sum** SUM computes the sum of the data in the AO.
-  **svd** SVD overloads the determinant function for Analysis objects.
-  **tan** TAN overloads the tan method for Analysis objects.
-  **timedomainfit** TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
-  **times** TIMES implements times operator for analysis objects.
-  **timeshift** TIMESHIFT for AO/tsdata objects, shifts the time axis such that $x(1) = 0$.
-  **transpose** TRANSPOSE overloads the '.' operator for Analysis Objects.
-  **uminus** UMINUS overloads the uminus operator for Analysis objects.
-  **upsample** UPSAMPLE overloads upsample function for AOs.
-  **var** VAR computes the variance of the data in the AO.
-  **xcorr** XCORR makes cross-correlation estimates of the time-series
-  **xspect** XSPEC performs cross-spectral analysis of various forms.
-  **zeropad** ZEROPAD zero pads the input data series.

SOURCE CODE [^]

```

0001 % COPY makes a (deep) copy of the input AOs.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY makes a deep copy of the input AOs.
0005 %
0006 % CALL:          b = copy(a, flag)
0007 %
0008 % INPUTS:        a    - input analysis object
0009 %                flag - 1: make a deep copy, 0: return copies of handles
0010 %
0011 % OUTPUTS:       b - copy of inputs
0012 %
0013 % This is a transparent function and adds no history.
0014 %
0015 % VERSION:       $Id: copy.m,v 1.14 2008/09/03 16:29:33 hewitson Exp $
0016 %
0017 % HISTORY:       14-07-2008 M Hewitson
0018 %                Creation
0019 %

```

```
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = copy(old, deepcopy)
0023
0024     if deepcopy
0025         t0 = time;
0026         prov = provenance;
0027         % Loop over input AOs
0028         s = size(old);
0029         obj(s(1),s(2)) = ao;
0030         for kk=1:numel(old)
0031             utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
0032             % New defaults for all these objects
0033             % copy properties
0034             if ~isempty(old(kk).data)
0035                 obj(kk).data = copy(old(kk).data, 1);
0036             end
0037             obj(kk).mfile = old(kk).mfile;
0038             obj(kk).mfilename = old(kk).mfilename;
0039             obj(kk).mdlfile = old(kk).mdlfile;
0040             obj(kk).plotinfo = copy(old(kk).plotinfo, 1);
0041             obj(kk).description = old(kk).description;
0042             obj(kk).hist = old(kk).hist;
0043             obj(kk).name = old(kk).name;
0044             % These are reset on copy
0045             obj(kk).procinfo = plist;
0046             obj(kk).created = t0;
0047             obj(kk).creator = prov;
0048             obj(kk).version = ao.VEROUT;
0049         end
0050     else
0051         obj = old;
0052     end
0053     varargout{1} = obj;
0054 end
0055
0056
```

demux

PURPOSE ^

DEMUX splits the input vector of AOs into a number of output AOs.

SYNOPSIS ^

function varargout = demux(varargin)

DESCRIPTION ^

```
DEMUX splits the input vector of AOs into a number of output AOs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DEMUX splits the input vector of AOs into a number of output AOs.
REMARK:      This is a transparent function and as such does not add history.
CALL:        [a1,a2,...] = demux(as)
VERSION:      $Id: demux.m,v 1.13 2008/09/05 11:05:29 ingo Exp $
M-FILE INFO:  Get information about this methods by calling
               >> ao.getInfo('demux')
               Get information about a specified set-plist by calling:
               >> ao.getInfo('demux', 'None')
HISTORY:      28-03-07 M Hewitson
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  `function ii = getInfo(varargin)`
-  `function pl = getDefaultPlist()`

SOURCE CODE ^

```
0001 % DEMUX splits the input vector of AOs into a number of output AOs.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DEMUX splits the input vector of AOs into a number of output AOs.
0005 %
0006 % REMARK:      This is a transparent function and as such does not add history.
0007 %
0008 % CALL:        [a1,a2,...] = demux(as)
0009 %
0010 % VERSION:      $Id: demux.m,v 1.13 2008/09/05 11:05:29 ingo Exp $
0011 %
0012 % M-FILE INFO:  Get information about this methods by calling
0013 %               >> ao.getInfo('demux')
0014 %
0015 %               Get information about a specified set-plist by calling:
0016 %               >> ao.getInfo('demux', 'None')
0017 %
0018 % HISTORY:      28-03-07 M Hewitson
```

```

0019 %                               Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = demux(varargin)
0024 %
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033 %
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037 %
0038 % Collect all AOs
0039 as = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040 %
0041 % get number of input AOs
0042 n = numel(as);
0043 if nargout > n
0044     error('### too many output arguments');
0045 elseif nargout < n
0046     error('### not enough output arguments');
0047 end
0048 %
0049 for j=1:nargout
0050     % map to output AO
0051     varargout{j} = as(j);
0052 end
0053 end
0054 %
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059 %
0060 if nargin == 1 && strcmpi(varargin{1}, 'None')
0061     sets = {};
0062     pl = [];
0063 else
0064     sets = {'Default'};
0065     pl = getDefaultPlist;
0066 end
0067 % Build info object
0068 ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: demux.m,v 1.13
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %-----
0072 % Get Default Plist
0073 %-----
0074 function pl = getDefaultPlist()
0075     pl = plist();
0076 end
0077

```


find

PURPOSE ^

FIND particular samples that satisfy the input query and return a new AO.

SYNOPSIS ^

function varargout = find(varargin)

DESCRIPTION ^

```
FIND particular samples that satisfy the input query and return a new AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FIND particular samples that satisfy the input query and return a new AO.

CALL:      b = find(a, 'x > 10 & x < 100')
           b = find(a, plist)

PARAMETERS: query - a string specifying a query on 'x' or 'y' for axes data
              or 'vals' for cdata

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('find')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('find', 'None')

VERSION:    $Id: find.m,v 1.22 2008/09/05 11:05:29 ingo Exp $

HISTORY:    14-05-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [t0](#) T0 Get the data property 't0'.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [attachm](#) ATTACHM attach an m file to the analysis object.
-  [compute](#) COMPUTE performs the given operations on the input AOs.
-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.
-  [ctranspose](#) CTRANSPOSE overloads the ' operator for Analysis Objects.
-  [delay](#) DELAY delays a time-series using various methods.
-  [detrend](#) DETREND detrends the input analysis object using a polynomial of degree N.
-  [dft](#) DFT computes the DFT of the input time-series at the requested frequencies.
-  [diff](#) DIFF differentiates the data in AO.
-  [dopplercorr](#) FOO description for function 'foo' in one line. Necessary for lookfor functionality.

-  [downsample](#) DOWNSAMPLE AOs containing time-series data.
-  [dropduplicates](#) DROPDUPLICATES drops all duplicate samples in time-series AOs.
-  [dsmean](#) DSMEAN performs a simple downsampling by taking the mean of every N samples.
-  [export](#) EXPORT export an analysis object to a text file.
-  [fft](#) FFT overloads the fft method for Analysis objects.
-  [filter](#) FILTER overrides the filter function for analysis objects.
-  [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [firwhiten](#) FIRWHITEN whitens the input time-series by building an FIR whitening filter.
-  [fixfs](#) FIXFS resamples the input time-series to have a fixed sample rate.
-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input PSD.
-  [fromDataInMAT](#) FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object
-  [fromDatafile](#) FROMDATAFILE Construct an ao from filename AND parameter list
-  [fromFSfcn](#) FROMFSFCN Construct an ao from a fs-function string
-  [fromFcn](#) FROMFCN Construct an ao from a function string
-  [fromPolyval](#) FROMPOLYVAL Construct a time-series ao from polynomial coefficients
-  [fromPzmodel](#) FROMPZMODEL Construct a time-series ao from polynomial coefficients
-  [fromSpecWin](#) FROMSPECWIN Construct an ao from a Spectral window
-  [fromTSfcn](#) FROMTSFCN Construct an ao from a ts-function string
-  [fromVals](#) FROMVALS Construct an ao from a value set
-  [fromWaveform](#) FROMWAVEFORM Construct an ao from a waveform
-  [gapfilling](#) GAPFILLING fills possible gaps in data.
-  [hist](#) HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
-  [ifft](#) IFFT overloads the ifft operator for Analysis objects.
-  [index](#) INDEX index into an AO array or matrix. This properly captures the history.
-  [interp](#) INTERP interpolate the values in the input AO(s) at new values.
-  [interpmissing](#) INTERPMISSING interpolate missing samples in a time-series.
-  [iplot](#) IPLOT provides an intelligent plotting tool for LTPDA.
-  [join](#) JOIN multiple AOs into a single AO.
-  [lincom](#) LINCOM
-  [linedetect](#) LINEDETECT find spectral lines in the ao/fsdata objects.
-  [lpsd](#) LPSD implement the LPSD algorithm for analysis objects.
-  [lxspecc](#) LXSPEC performs log-scale cross-spectral analysis of various forms.
-  [max](#) MAX computes the maximum value of the data in the AO.
-  [mdc1_ifo2acc_fd](#) MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
-  [mdc1_ifo2acc_inloop](#) MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.
-  [mdc1_input_noises](#) LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.
-  [mdc1_x2acc](#) MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter
-  [min](#) MIN computes the minimum value of the data in the AO.
-  [polyfit](#) POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
-  [psd](#) PSD makes power spectral density estimates of the time-series objects
-  [resample](#) RESAMPLE overloads resample function for AOs.
-  [search](#) SEARCH selects AOs that match the given name.
-  [select](#) SELECT select particular samples from the input AOs and return new AOs with only those samples.
-  [setDescription](#) SETDESCRIPTION sets the 'description' property of the ao.
-  [setFs](#) SETFS sets the 'fs' property of the ao.
-  [setT0](#) SETT0 sets the 't0' property of the ao.
-  [setX](#) SETX sets the 'x' property of the ao.

-  **setXunits** SETXUNITS sets the 'xunits' property of the ao.
-  **setY** SETY sets the 'y' property of the ao.
-  **setYunits** SETYUNITS sets the 'yunits' property of the ao.
-  **setZ** SETZ sets the 'z' property of the ao.
-  **smoother** SMOOTHER smooths a given series of data points using the specified method.
-  **sort** SORT the values in the AO.
-  **spectrogram** SPECTROGRAM computes a spectrogram of the given ao/tsdata.
-  **spikecleaning** spikecleaning detects and corrects possible spikes in analysis objects
-  **split** SPLIT split an analysis object into the specified segments.
-  **sqrt** SQRT computes the square root of the data in the AO.
-  **transpose** TRANSPPOSE overloads the '.' operator for Analysis Objects.
-  **upsample** UPSAMPLE overloads upsample function for AOs.
-  **validate** VALIDATE checks that the input Analysis Object is reproducible and valid.
-  **welch** WELCH Welch spectral estimation method.
-  **xcorr** XCORR makes cross-correlation estimates of the time-series
-  **xspec** XSPEC performs cross-spectral analysis of various forms.
-  **zeropad** ZEROPAD zero pads the input data series.

SUBFUNCTIONS [^]

-  **function** ii = getInfo(varargin)
-  **function** pl_default = getDefaultPlist()

SOURCE CODE [^]

```

0001 % FIND particular samples that satisfy the input query and return a new AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FIND particular samples that satisfy the input query and return a new
AO.
0005 %
0006 % CALL:          b = find(a, 'x > 10 & x < 100')
0007 %              b = find(a, plist)
0008 %
0009 % PARAMETERS:   query - a string specifying a query on 'x' or 'y' for axes data
0010 %              or 'vals' for cdata
0011 %
0012 % M-FILE INFO:  Get information about this methods by calling
0013 %              >> ao.getInfo('find')
0014 %
0015 %              Get information about a specified set-plist by calling:
0016 %              >> ao.getInfo('find', 'None')
0017 %
0018 % VERSION:      $Id: find.m,v 1.22 2008/09/05 11:05:29 ingo Exp $
0019 %
0020 % HISTORY:      14-05-07 M Hewitson
0021 %              Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = find(varargin)
0026
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 import utils.const.*
0034 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0035
0036 % Collect input variable names
0037 in_names = cell(size(varargin));
0038 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0039
0040 % Collect all AOs and plists
0041 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042 [pl, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0043

```

```

0044 % Decide on a deep copy or a modify
0045 bs = copy(as, nargout);
0046
0047 % combine plists
0048 pl = combine(pl, getDefaultPlist());
0049
0050 % initialise queries
0051 queries = '';
0052 % extract any from inputs
0053 for j=1:numel(rest)
0054     if ischar(rest{j})
0055         queries = [queries ' & ' rest{j}];
0056     end
0057 end
0058
0059 % Get sample selection from plist
0060 queries = [queries ' & ' find(pl, 'query')];
0061
0062 % We may have pre or trailing '&' now
0063 queries = strtrim(queries);
0064 if queries(1) == '&'
0065     queries = queries(2:end);
0066 end
0067 if queries(end) == '&'
0068     queries = queries(1:end-1);
0069 end
0070
0071 % Loop over input AOs
0072 for j=1:numel(bs)
0073
0074     % The user uses the variables x and y so we need
0075     % to put them in the workspace.
0076     x = bs(j).data.getX;
0077     y = bs(j).data.getY;
0078
0079     idx = [];
0080     cmd = sprintf('idx = find(%s);', queries);
0081     try
0082         eval(cmd);
0083     catch ME
0084         error('%s\n\n### please use x or y for the queries.', ME.message);
0085     end
0086
0087     % set data
0088     bs(j).data.setXY(x(idx), y(idx));
0089
0090     if isa(bs(j).data, 'tsdata')
0091         % Check fs and see if is x evenly sampled?
0092         [fs,t0,fitted] = tsdata.fitfs(x(idx));
0093         if fitted
0094             bs(j).data.setXY(x(idx), y(idx));
0095             bs(j).data.setFs(fs);
0096         else
0097             % drop x
0098             bs(j).data.setY(y(idx));
0099             bs(j).data.collapseX;
0100         end
0101     end
0102
0103     % set name
0104     bs(j).setName(sprintf('select(%s)', ao_invars{j}), 'internal');
0105     % add history
0106     bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0107 end
0108
0109 % Set output
0110 varargin{1} = bs;
0111 end
0112
0113 %-----
0114 % Get Info Object
0115 %-----
0116 function ii = getInfo(varargin)
0117     if nargin == 1 && strcmpi(varargin{1}, 'None')
0118         sets = {};
0119         pls = [];
0120     else
0121         sets = {'Default'};
0122         pls = getDefaultPlist;
0123     end
0124     % Build info object
0125     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: find.m,v 1.22
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0126 end
0127
0128 %-----

```

```
0129 % Get Default Plist
0130 %-----
0131 function pl_default = getDefaultPlist()
0132     pl_default = plist('query', '');
0133 end
0134
0135
```

index

PURPOSE ^

INDEX index into an AO array or matrix. This properly captures the history.

SYNOPSIS ^

function varargout = index(varargin)

DESCRIPTION ^

```
INDEX index into an AO array or matrix. This properly captures the history.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: INDEX into an AO array or matrix. This properly captures the
              history. This function is also called when standard () indexing
              is used for AO arrays.

CALL:        b = index(a, i)
              b = index(a, i, j)

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('index')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('index', 'None')

VERSION:     $Id: index.m,v 1.17 2008/09/05 11:05:29 ingo Exp $

HISTORY:     02-05-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % INDEX index into an AO array or matrix. This properly captures the history.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: INDEX into an AO array or matrix. This properly captures the
0005 %               history. This function is also called when standard () indexing
0006 %               is used for AO arrays.
0007 %
0008 % CALL:        b = index(a, i)
0009 %               b = index(a, i, j)
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 %               >> ao.getInfo('index')
```

```

0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> ao.getInfo('index', 'None')
0016 %
0017 % VERSION:      $Id: index.m,v 1.17 2008/09/05 11:05:29 ingo Exp $
0018 %
0019 % HISTORY:      02-05-07 M Hewitson
0020 %                Creation
0021 %
0022 %%%%%%%%%%%
0023 %
0024 %
0025 function varargout = index(varargin)
0026
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 if nargin == 0
0034     error('### index cannot be used as a modifier. Please give an output variable.');
```

```

0035 end
0036
0037 import utils.const.*
0038 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0039
0040 % We can only effectively handle a single matrix of AOs
0041 af = 0;
0042 for j=1:nargin
0043     if isa(varargin{j}, 'ao')
0044         af = af + 1;
0045     end
0046 end
0047 if af > 1
0048     error('### Input a single matrix of AOs for indexing.');
```

```

0049 end
0050
0051 % Collect input variable names
0052 in_names = cell(size(varargin));
0053 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0054
0055 % Collect all AOs and plists
0056 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0057 [pl, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0058
0059 % Decide on a deep copy or a modify
0060 bs = copy(as, nargin);
0061
0062 % Combine plists
0063 pl = combine(pl, getDefaultPlist);
0064
0065 % Get indices
0066 idxi = find(pl, 'i');
0067 idxj = find(pl, 'j');
```

```

0068 if isempty(idxi) || isempty(idxj)
0069     % go through the other inputs
0070     vals = [];
0071     for j=1:numel(rest)
0072         if isnumeric(rest{j}) && numel(rest{j})==1
0073             vals = [vals rest{j}];
0074         end
0075     end
0076     if ~isempty(vals)
0077         idxi = vals(1);
0078         if numel(vals) > 1
0079             idxj = vals(2);
0080         end
0081     end
0082 end
0083
0084 if isempty(idxi) && isempty(idxj)
0085     error('### Please provide an index of index pair.');
```

```

0086 end
0087
0088 % Gather the input histories
0089 inhists = [];
0090 for j=1:numel(bs)
0091     inhists = [inhists bs(j).hist];
0092 end
0093
0094 % Now index with either (i) or (i,j)
0095 if isempty(idxj)
0096     aout = bs(idxi);
0097     inhists = inhists(idxi);
0098 else

```

```

0099     aout = bs(idxi, idxj);
0100     inhists = inhists(sub2ind(size(bs), idxi, idxj));
0101 end
0102
0103 % Make sure we store the indices
0104 pl = pl.pset('i', idxi);
0105 pl = pl.pset('j', idxj);
0106
0107 % Name and add history to all outputs
0108 for j=1:numel(aout)
0109     if isempty(idxi)
0110         aout(j).setName(sprintf('%s(%d)', ao_invars{1}, idxi), 'internal');
0111     else
0112         aout(j).setName(sprintf('%s(%d,%d)', ao_invars{1}, idxi, idxj), 'internal');
0113     end
0114     % add history
0115     aout(j).addHistory(getInfo, pl, ao_invars, inhists(j));
0116 end
0117
0118 % Set output
0119 varargout{1} = aout;
0120 end
0121
0122 %-----
0123 % Get Info Object
0124 %-----
0125 function ii = getInfo(varargin)
0126     if nargin == 1 && strcmpi(varargin{1}, 'None')
0127         sets = {};
0128         pl = [];
0129     else
0130         sets = {'Default'};
0131         pl = getDefaultPlist;
0132     end
0133     % Build info object
0134     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: index.m,v 1.17
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0135 end
0136
0137 %-----
0138 % Get Default Plist
0139 %-----
0140 function pl_default = getDefaultPlist()
0141     pl_default = plist('i', [], 'j', []);
0142 end
0143

```


join

PURPOSE ^

JOIN multiple AOs into a single AO.

SYNOPSIS ^

function varargout = join(varargin)

DESCRIPTION ^

```
JOIN multiple AOs into a single AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: JOIN multiple AOs into a single AO.
              If any two AOs overlap, then the values from the first appear
              in the output AO.

CALL:        b = join(a1,a2,pl)

REMARK:      Input AOs should be of the same type; if not, only AOs of the
              type of the first input AO will be joined together to produce
              the output.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('join')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('join', 'None')

VERSION:     $Id: join.m,v 1.24 2008/09/05 11:05:29 ingo Exp $

HISTORY:     03-06-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [max](#) MAX computes the maximum value of the data in the AO.
-  [min](#) MIN computes the minimum value of the data in the AO.
-  [sort](#) SORT the values in the AO.
-  [t0](#) T0 Get the data property 't0'.
-  [x](#) X Get the data property 'x'.
-  [xunits](#) XUNITS Get the data property 'xunits'.
-  [yunits](#) YUNITS Get the data property 'yunits'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)
-  [function Toff = getToff\(as\)](#)

SOURCE CODE ^

```

0001 % JOIN multiple AOs into a single AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: JOIN multiple AOs into a single AO.
0005 %             If any two AOs overlap, then the values from the first appear
0006 %             in the output AO.
0007 %
0008 % CALL:       b = join(a1,a2,pl)
0009 %
0010 % REMARK:     Input AOs should be of the same type; if not, only AOs of the
0011 %             type of the first input AO will be joined together to produce
0012 %             the output.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> ao.getInfo('join')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> ao.getInfo('join', 'None')
0019 %
0020 % VERSION:    $Id: join.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
0021 %
0022 % HISTORY:    03-06-07 M Hewitson
0023 %             Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 function varargout = join(varargin)
0028
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 if nargin == 0
0036     error('### join cannot be used as a modifier. Please give an output variable.');
```

```

0037 end
0038
0039 import utils.const.*
0040 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0041
0042 % Collect input variable names
0043 in_names = cell(size(varargin));
0044 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0045
0046 % Collect all AOs
0047 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0048
0049 %-----
0050 % Get data type from the first AO
0051 dtype = class(as(1).data);
0052
0053 %-----
0054 % Go through each AO and collect the data of type 'dtype'
0055
0056 histin = [];
0057 xo      = [];
0058 yo      = [];
0059 fs      = -1;
0060 dname    = '';
0061 aname    = '';
0062 xunits   = as(1).data.xunits;
0063 yunits   = as(1).data.yunits;
0064
0065 % Compute time offset for tsdata objects
0066 % to avoid rounding errors later
0067
0068 Toff = getToff(as);
0069 fsDifferent = false;
0070 % loop over AOs
0071 for j=1:numel(as)
0072     % Only get the data type we want
0073     if isa(as(j).data, dtype)
0074         switch dtype
0075             case 'tsdata'
0076                 % here we concatenate time-series
0077                 t0 = (as(j).data.t0.utc_epoch_milli - Toff)/1000;
0078                 % make proper time vector
0079                 x = as(j).data.getx + t0;
0080                 % only add samples past the end of existing
0081                 if isempty(xo)

```

```

0082         yo = as(j).data.getY;
0083         xo = x;
0084     else
0085         idxPost = find(x > max(xo));
0086         idxPre = find(x < min(xo));
0087         xo = [x(idxPre); xo; x(idxPost)];
0088         yo = [as(j).data.getY(idxPre); yo; as(j).data.getY(idxPost)];
0089     end
0090
0091     % store fs
0092     if fs>0 && as(j).data.fs ~= fs
0093         error('!!! Data has different sample rates !!!');
0094     end
0095     fs = as(j).data.fs;
0096
0097     % store xunits
0098     if xunits ~= as(j).data.xunits
0099         error('!!! Data has different X units !!!');
0100     end
0101     xunits = as(j).data.xunits;
0102
0103     % store yunits
0104     if yunits ~= as(j).data.yunits
0105         error('!!! Data has different Y units !!!');
0106     end
0107     yunits = as(j).data.yunits;
0108
0109     case 'fsdata'
0110         error('### I only work with time-series data at the moment.');
```

```

0111     case 'xydata'
0112         error('### I only work with time-series data at the moment.');
```

```

0113     case 'cdata'
0114         error('### I only work with time-series data at the moment.');
```

```

0115     otherwise
0116         error('### Unknown data type');
```

```

0117     end
0118     % Collect this input history
0119     histin = [histin as(j).hist];
0120     % Collect names, invars
0121     if ~isempty(aname)
0122         if ~strcmp(aname, as(j).name)
0123             aname = [aname ', ' as(j).name];
0124         end
0125     else
0126         aname = as(j).name;
0127     end
0128     else
0129         warning('!!! Ignoring AO input with data type %s', dtype);
0130     end
0131 end
0132 % Tidy up names
0133 dname = dname(2:end);
0134
0135 %-----
0136 % Now sort output vectors
0137 [xos, idx] = sort(xo);
0138 yos = yo(idx);
0139
0140 %% Build output data object
0141 switch dtype
0142     case 'tsdata'
0143         % get t0
0144         t0 = xos(1);
0145         xos = xos - t0;
0146         data = tsdata(xos, yos, fs);
0147         data.setT0((Toff+t0*1000));
0148         data.setXunits(xunits);
0149         data.setYunits(yunits);
0150     case 'fsdata'
0151     case 'xydata'
0152     case 'cdata'
0153 end
0154
0155 %-----
0156 % Build output AO
0157 a = ao(data);
0158 a.setName(aname, 'internal');
0159 % add history
0160 a.addHistory(getInfo, plist, ao_invars, histin);
0161
0162 %-----
0163 % set output
0164 varargout{1} = a;
0165 end
0166
0167 %-----

```

```

0168 % Get Info Object
0169 %-----
0170 function ii = getInfo(varargin)
0171     if nargin == 1 && strcmpi(varargin{1}, 'None')
0172         sets = {};
0173         pl = [];
0174     else
0175         sets = {'Default'};
0176         pl = getDefaultPlist;
0177     end
0178     % Build info object
0179     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: join.m,v 1.24
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0180 end
0181
0182 %-----
0183 % Get Default Plist
0184 %-----
0185 function pl = getDefaultPlist()
0186     pl = plist();
0187 end
0188
0189 %-----
0190 % Get Offset of this set of time-vectors
0191 %-----
0192 function Toff = getToff(as)
0193     Toff = 1e20;
0194     for j=1:length(as)
0195         if isa(as(j).data, 'tsdata')
0196             t0 = as(j).data.t0.utc_epoch_milli;
0197             if t0 < Toff
0198                 Toff = t0;
0199             end
0200         end
0201     end
0202 end
0203

```

len

PURPOSE ^

LEN overloads the length operator for Analysis objects. Length of the data samples.

SYNOPSIS ^

function varargout = len(varargin)

DESCRIPTION ^

```
LEN overloads the length operator for Analysis objects. Length of the data samples.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LEN overloads the length operator for Analysis objects.
              Number of data samples in the analysis object.
              LEN(ao) is the length of the elements of ao.data.

CALL:        l = len(ao_in);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('len')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('len', 'None')

VERSION:     $Id: len.m,v 1.19 2008/09/05 11:05:29 ingo Exp $

HISTORY:     24-04-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [findShortestVector](#) findShortestVector Returns the length of the shortest vector in samples
-  [lxspecc](#) LXSPEC performs log-scale cross-spectral analysis of various forms.
-  [xcorr](#) XCORR makes cross-correlation estimates of the time-series
-  [xspecc](#) XSPEC performs cross-spectral analysis of various forms.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % LEN overloads the length operator for Analysis objects. Length of the data samples.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LEN overloads the length operator for Analysis objects.
0005 %              Number of data samples in the analysis object.
0006 %              LEN(ao) is the length of the elements of ao.data.
0007 %
0008 % CALL:        l = len(ao_in);
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
```

```

0011 %                >> ao.getInfo('len')
0012 %
0013 %                Get information about a specified set-plist by calling:
0014 %                >> ao.getInfo('len', 'None')
0015 %
0016 % VERSION:      $Id: len.m,v 1.19 2008/09/05 11:05:29 ingo Exp $
0017 %
0018 % HISTORY:      24-04-07 M Hewitson
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%
0022 %
0023 function varargout = len(varargin)
0024 %
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 if nargin == 0
0032     error('### len cannot be used as a modifier. Please give an output variable.');
```

```

0033 end
0034 %
0035 import utils.const.*
0036 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0037 %
0038 % Collect input variable names
0039 in_names = cell(size(varargin));
0040 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0041 %
0042 % Collect all AOs and plists
0043 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0044 %
0045 % go through analysis objects
0046 data_len = zeros(size(as));
0047 for j=1: numel(as)
0048     data_len(j) = length(as(j).data.getY);
0049 end
0050 %
0051 % Set outputs
0052 varargout{1} = data_len;
0053 end
0054 %
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl = [];
0062     else
0063         sets = {'Default'};
0064         pl = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: len.m,v 1.19
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0068 end
0069 %
0070 %-----
0071 % Get Default Plist
0072 %-----
0073 function pl_default = getDefaultPlist()
0074     pl_default = plist();
0075 end
0076 %
0077 % END

```

md5

PURPOSE ^

MD5 computes an MD5 checksum from an analysis objects.

SYNOPSIS ^

function varargout = md5(varargin)

DESCRIPTION ^

```
MD5 computes an MD5 checksum from an analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MD5 computes an MD5 checksum from an analysis objects.

CALL:      h = md5(a)

INPUTS:    a - input analysis object

OUTPUTS:   h - md5 hash

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('md5')
              Get information about a specified set-plist by calling:
              >> ao.getInfo('md5', 'None')

VERSION:   $Id: md5.m,v 1.12 2008/09/05 11:05:29 ingo Exp $

HISTORY:   15-09-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MD5 computes an MD5 checksum from an analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MD5 computes an MD5 checksum from an analysis objects.
0005 %
0006 % CALL:      h = md5(a)
0007 %
0008 % INPUTS:    a - input analysis object
0009 %
0010 % OUTPUTS:   h - md5 hash
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> ao.getInfo('md5')
```

```

0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> ao.getInfo('md5', 'None')
0017 %
0018 % VERSION:      $Id: md5.m,v 1.12 2008/09/05 11:05:29 ingo Exp $
0019 %
0020 % HISTORY:      15-09-2007 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%
0024
0025 function varargout = md5(varargin)
0026
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 import utils.const.*
0034 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0035
0036 % Collect input variable names
0037 in_names = cell(size(varargin));
0038 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0039
0040 % Collect all AOs and plists
0041 as = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042
0043 h = {};
0044 for ii = 1:numel(as)
0045
0046     %%%%%%%%%%% Convert object to XML
0047     % make pointer to xml document
0048     xml = com.mathworks.xml.XMLUtils.createDocument('ltpda_object');
0049     % extract parent node
0050     parent = xml.getDocumentElement;
0051     % write obj into xml
0052     utils.helper.xmlwrite(as(ii), xml, parent, '');    % Save the XML document.
0053
0054     h = [h cellstr(utils.prog.hash(xmlwrite(xml), 'MD5'))];
0055
0056 end
0057
0058 % Set outputs
0059 h = reshape(h, size(as));
0060 if numel(h) == 1
0061     varargout{1} = cell2mat(h);
0062 else
0063     varargout{1} = h;
0064 end
0065 end
0066
0067 %-----
0068 % Get Info Object
0069 %-----
0070 function ii = getInfo(varargin)
0071     if nargin == 1 && strcmpi(varargin{1}, 'None')
0072         sets = {};
0073         pl = [];
0074     else
0075         sets = {'Default'};
0076         pl = getDefaultPlist;
0077     end
0078     % Build info object
0079     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: md5.m,v 1.12
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0080 end
0081
0082 %-----
0083 % Get Default Plist
0084 %-----
0085 function pl = getDefaultPlist()
0086     pl = plist();
0087 end
0088

```

search

PURPOSE ^

SEARCH selects AOs that match the given name.

SYNOPSIS ^

```
function varargout = search(varargin)
```

DESCRIPTION ^

```
SEARCH selects AOs that match the given name.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SEARCH selects AOs that match the given name using a regular
expression (help regexp).

CALL:      b = search(a, 'foo') % get all AOs from <a> called 'foo'
           b = search(a, 'foo*') % get all AOs from <a> with a name like 'foo'
           b = search(a, pl)

PARAMETERS: 'regexp' - a string specifying the regular expression

This function returns the handles of the AOs that match the regular
expression. No object copying is done.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('search')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('search', 'None')

VERSION:     $Id: search.m,v 1.4 2008/09/05 11:05:29 ingo Exp $

HISTORY:     10-07-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SEARCH selects AOs that match the given name.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SEARCH selects AOs that match the given name using a regular
0005 %               expression (help regexp).
0006 %
0007 % CALL:      b = search(a, 'foo') % get all AOs from <a> called 'foo'
```



```

0008 %           b = search(a, 'foo*') % get all AOs from <a> with a name like 'foo'
0009 %           b = search(a, pl)
0010 %
0011 % PARAMETERS: 'regex' - a string specifying the regular expression
0012 %
0013 % This function returns the handles of the AOs that match the regular
0014 % expression. No object copying is done.
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 %               >> ao.getInfo('search')
0018 %
0019 %               Get information about a specified set-plist by calling:
0020 %               >> ao.getInfo('search', 'None')
0021 %
0022 % VERSION:      $Id: search.m,v 1.4 2008/09/05 11:05:29 ingo Exp $
0023 %
0024 % HISTORY:      10-07-08 M Hewitson
0025 %               Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = search(varargin)
0030
0031 % Check if this is a call for parameters
0032 if utils.helper.isinfocall(varargin{:})
0033     varargout{1} = getInfo(varargin{3});
0034     return
0035 end
0036
0037 import utils.const.*
0038 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0039
0040 % Collect input variable names
0041 in_names = cell(size(varargin));
0042 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0043
0044 % Collect all AOs and plists
0045 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0046 [pl, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0047
0048 % Build a cell array of the input AO names
0049 aonames = {};
0050 for j=1:numel(as)
0051     aonames = [aonames {as(j).name}];
0052 end
0053
0054 %----- Get expression
0055 % first look in direct inputs
0056 exp = '';
0057 for j=1:numel(rest)
0058     if ischar(rest{j})
0059         exp = rest{j};
0060         break;
0061     end
0062 end
0063 % then in plist
0064 if isempty(exp)
0065     exp = find(pl, 'regex');
0066 end
0067 if isempty(exp)
0068     error('### Please specify an expression to match.');
```

```
0094     sets = {'Default'};
0095     pl    = getDefaultPlist;
0096 end
0097 % Build info object
0098 ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: search.m,v 1.4
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0099 end
0100
0101 %-----
0102 % Get Default Plist
0103 %-----
0104 function pl_default = getDefaultPlist()
0105     pl_default = plist('regexp', '');
0106 end
0107
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setDescription

PURPOSE ^

SETDESCRIPTION sets the 'description' property of the ao.

SYNOPSIS ^

function varargout = setDescription(varargin)

DESCRIPTION ^

```
SETDESCRIPTION sets the 'description' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETDESCRIPTION sets the 'description' property of the ao.

CALL:      ao = setDescription(ao, val)
           obj = obj.setDescription(plist('description', 'bla bla'));

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setDescription')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setDescription', 'None')

VERSION:    $Id: setDescription.m,v 1.7 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY makes a (deep) copy of the input AOs.
-  **find** FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  **ao** AO analysis object class constructor.

SUBFUNCTIONS ^

-  **function ii = getInfo(varargin)**
-  **function pl = getDefaultPlist()**

SOURCE CODE ^

```
0001 % SETDESCRIPTION sets the 'description' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETDESCRIPTION sets the 'description' property of the ao.
0005 %
0006 % CALL:      ao = setDescription(ao, val)
0007 %           obj = obj.setDescription(plist('description', 'bla bla'));
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('setDescription')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setDescription', 'None')
0014 %
0015 % VERSION:      $Id: setDescription.m,v 1.7 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = setDescription(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032
0033     %%% decide whether we modify the first object, or create a new one.
0034     varargin{1} = copy(varargin{1}, nargout);
0035
0036     varargin{1}.description = varargin{2};
0037     varargout{1} = varargin{1};
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all AOs
0049 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051
0052 %%% If pls contains only one plist with the only key 'description' then set the
0053 %%% property with a plist.
0054 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls,
'description')
0055     rest{1} = find(pls, 'description');
0056 end
0057
0058 if numel(rest) ~= 1
0059     error('### Please specify a value for description, either in a plist or
directly.');
```

```

0060 end
0061
0062 %%% Combine plists
0063 if isempty(pls)
0064     pls = plist('description', rest{1});
0065 else
0066     pls = pls.combine(plist('description', rest{1}));
0067 end
0068
0069 % Decide on a deep copy or a modify
0070 bs = copy(as, nargout);
0071
0072 % Loop over AOs
0073 for j=1:numel(bs)
0074     bs(j).description = rest{1};
0075     bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
0076 end
0077
0078 if nargout > 0
0079     varargout{1} = bs;
0080 end
0081 end
0082
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084 %
0085 % Local Functions
0086 %-----
0087 % Get Info Object
0088 %-----
0089 function ii = getInfo(varargin)
0090
0091 if nargin == 1 && strcmpi(varargin{1}, 'None')
0092     sets = {};
0093     pl = [];
0094 else
0095     sets = {'Default'};
```

```
0096     pl    = getDefaultPlist;
0097     end
0098     % Build info object
0099     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id:
setDescription.m,v 1.7 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106     pl = plist('description', '');
0107 end
0108
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setFs

PURPOSE ^

SETFS sets the 'fs' property of the ao.

SYNOPSIS ^

function varargout = setFs(varargin)

DESCRIPTION ^

```
SETFS sets the 'fs' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETFS sets the 'fs' property of the ao.

CALL:      obj = obj.setFs(val)
           obj = obj.setFs(plist('fs', 123);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setFs')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setFs', 'None')

VERSION:    $Id: setFs.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETFS sets the 'fs' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETFS sets the 'fs' property of the ao.
0005 %
0006 % CALL:      obj = obj.setFs(val)
0007 %           obj = obj.setFs(plist('fs', 123);
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('setFs')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setFs', 'None')
0014 %
0015 % VERSION:      $Id: setFs.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = setFs(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032
0033     %%% decide whether we modify the first object, or create a new one.
0034     varargin{1} = copy(varargin{1}, nargout);
0035
0036     varargin{1}.data.setFs(varargin{2});
0037     varargout{1} = varargin{1};
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all AOs
0049 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051
0052 %%% If pls contains only one plist with the only key 'fs' then set the
0053 %%% property with a plist.
0054 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'fs')
0055     rest{1} = find(pls, 'fs');
0056 end
0057
0058 if numel(rest) ~= 1
0059     error('### Please specify a value for Fs, either in a plist or directly.');
```

```
0097 end
0098
0099 %-----
0100 % Get Default Plist
0101 %-----
0102 function pl = getDefaultPlist()
0103     pl = plist('fs', NaN);
0104     % END
0105 end
0106
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setT0

PURPOSE ^

SETT0 sets the 't0' property of the ao.

SYNOPSIS ^

function varargout = setT0(varargin)

DESCRIPTION ^

```
SETT0 sets the 't0' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETT0 sets the 't0' property of the ao.

CALL:      ao = setT0(ao, val)
           obj = obj.setT0(plist('t0', time(1234)));

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setT0')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setT0', 'None')

VERSION:    $Id: setT0.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETT0 sets the 't0' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETT0 sets the 't0' property of the ao.
0005 %
0006 % CALL:      ao = setT0(ao, val)
0007 %           obj = obj.setT0(plist('t0', time(1234)));
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('setT0')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setT0', 'None')
0014 %
0015 % VERSION:      $Id: setT0.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021 %
0022 function varargout = setT0(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032
0033     %%% decide whether we modify the first object, or create a new one.
0034     varargin{1} = copy(varargin{1}, nargout);
0035
0036     varargin{1}.data.setT0(varargin{2});
0037     varargout{1} = varargin{1};
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all AOs
0049 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051
0052 %%% If pls contains only one plist with the only key 't0' then set the
0053 %%% property with a plist.
0054 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 't0')
0055     rest{1} = find(pls, 't0');
0056 end
0057
0058 if numel(rest) ~= 1
0059     error('### Please specify a value for t0, either in a plist or directly.');
```

```
0098 % Build info object
0099 ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: setT0.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106     pl = plist('t0', time(0));
0107 end
0108
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setX

PURPOSE ^

SETX sets the 'x' property of the ao.

SYNOPSIS ^

function varargout = setX(varargin)

DESCRIPTION ^

```
SETX sets the 'x' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETX sets the 'x' property of the ao.

CALL:      ao = setX(ao, val)
           obj = obj.setX(plist('x', [1 2 3]));

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setX')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setX', 'None')

VERSION:    $Id: setX.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETX sets the 'x' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETX sets the 'x' property of the ao.
0005 %
0006 % CALL:      ao = setX(ao, val)
0007 %           obj = obj.setX(plist('x', [1 2 3]));
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %           >> ao.getInfo('setX')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setX', 'None')
0014 %
0015 % VERSION:      $Id: setX.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021 %
0022 function varargout = setX(varargin)
0023 %
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029 %
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032     %%% decide whether we modify the first object, or create a new one.
0033     varargin{1} = copy(varargin{1}, nargout);
0034     varargin{1}.data.setX(varargin{2});
0035     varargout{1} = varargin{1};
0036     return
0037 end
0038 %
0039 %
0040 import utils.const.*
0041 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0042 %
0043 % Collect input variable names
0044 in_names = cell(size(varargin));
0045 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0046 %
0047 % Collect all AOs
0048 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0049 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0050 %
0051 %%% If pls contains only one plist with the only key 'x' then set the
0052 %%% property with a plist.
0053 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'x')
0054     rest{1} = find(pls, 'x');
0055 end
0056 %
0057 if numel(rest) ~= 1
0058     error('### Please specify a value for x, either in a plist or directly.');
```

```
0098 % Build info object
0099 ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: setX.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106     pl = plist('x', []);
0107 end
0108
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setXY

PURPOSE ^

SETXY sets the 'xy' property of the ao.

SYNOPSIS ^

function varargout = setXY(varargin)

DESCRIPTION ^

```
SETXY sets the 'xy' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETXY sets the 'xy' property of the ao.

CALL:      ao = setXY(ao, x, y)
           obj = obj.setXY(plist('x', [1 2 3], 'y', [1 2 3]));

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setXY')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setXY', 'None')

VERSION:    $Id: setXY.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETXY sets the 'xy' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETXY sets the 'xy' property of the ao.
0005 %
0006 % CALL:      ao = setXY(ao, x, y)
0007 %           obj = obj.setXY(plist('x', [1 2 3], 'y', [1 2 3]));
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
```

```

0010 %                >> ao.getInfo('setXY')
0011 %
0012 %                Get information about a specified set-plist by calling:
0013 %                >> ao.getInfo('setXY', 'None')
0014 %
0015 % VERSION:      $Id: setXY.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021 %
0022 function varargout = setXY(varargin)
0023 %
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029 %
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032 %
0033 %%% decide whether we modify the first object, or create a new one.
0034 varargin{1} = copy(varargin{1}, nargout);
0035 %
0036 varargin{1}.data.setXY(varargin{2}, varargin{3});
0037 varargout{1} = varargin{1};
0038 return
0039 end
0040 %
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043 %
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047 %
0048 % Collect all AOs
0049 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051 %
0052 %%% Look for numeric values in rest
0053 x = [];
0054 y = [];
0055 if numel(rest) == 2 && isnumeric(rest{1}) && isnumeric(rest{2}) && numel(rest{1}) ==
numel(rest{2})
0056     x = rest{1};
0057     y = rest{2};
0058 end
0059 %
0060 %%% If pls contains parameters X and Y, get the values from there
0061 if isempty(x)
0062     x = pls.find('x');
0063 end
0064 if isempty(y)
0065     y = pls.find('y');
0066 end
0067 %
0068 if isempty(x) || isempty(y)
0069     error('### Please specify a value for X and Y, either in a plist or directly.');
```



```
0095 function ii = getInfo(varargin)
0096
0097     if nargin == 1 && strcmpi(varargin{1}, 'None')
0098         sets = {};
0099         pl   = [];
0100     else
0101         sets = {'Default'};
0102         pl   = getDefaultPlist;
0103     end
0104     % Build info object
0105     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: setXY.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0106 end
0107
0108 %-----
0109 % Get Default Plist
0110 %-----
0111 function pl = getDefaultPlist()
0112     pl = plist('x', [], 'y', []);
0113 end
0114
```

setXunits

PURPOSE ^

SETXUNITS sets the 'xunits' property of the ao.

SYNOPSIS ^

function varargout = setXunits(varargin)

DESCRIPTION ^

```
SETXUNITS sets the 'xunits' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETXUNITS sets the 'xunits' property of the ao.

CALL:      ao = setXunits(ao, val)
           obj = obj.setXunits(plist('xunits', 'sec'));

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setXunits')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setXunits', 'None')

VERSION:    $Id: setXunits.m,v 1.9 2008/09/05 14:15:33 hewitson Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETXUNITS sets the 'xunits' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETXUNITS sets the 'xunits' property of the ao.
0005 %
0006 % CALL:      ao = setXunits(ao, val)
0007 %           obj = obj.setXunits(plist('xunits', 'sec'));
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('setXunits')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setXunits', 'None')
0014 %
0015 % VERSION:      $Id: setXunits.m,v 1.9 2008/09/05 14:15:33 hewitson Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = setXunits(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032
0033     %%% decide whether we modify the first object, or create a new one.
0034     varargin{1} = copy(varargin{1}, nargout);
0035
0036     varargin{1}.data.setXunits(varargin{2});
0037     varargout{1} = varargin{1};
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all AOs
0049 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051
0052 %%% If pls contains only one plist with the only key 'xunits' then set the
0053 %%% property with a plist.
0054 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls,
'xunits')
0055     rest{1} = find(pls, 'xunits');
0056 end
0057
0058 if numel(rest) ~= 1
0059     error('### Please specify a value for xunits, either in a plist or directly.');
```

```
0097     end
0098     % Build info object
0099     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: setXunits.m,v 1.9
2008/09/05 14:15:33 hewitson Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106     pl = plist('xunits', unit);
0107 end
0108
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setY

PURPOSE ^

SETY sets the 'y' property of the ao.

SYNOPSIS ^

function varargout = setY(varargin)

DESCRIPTION ^

```
SETY sets the 'y' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETY sets the 'y' property of the ao.

CALL:      ao = setY(ao, val)
           ao = ao.setY(plist('y', 123);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setY')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setY', 'None')

VERSION:    $Id: setY.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETY sets the 'y' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETY sets the 'y' property of the ao.
0005 %
0006 % CALL:      ao = setY(ao, val)
0007 %           ao = ao.setY(plist('y', 123);
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('setY')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setY', 'None')
0014 %
0015 % VERSION:      $Id: setY.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = setY(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032     %%% decide whether we modify the first object, or create a new one.
0033     varargin{1} = copy(varargin{1}, nargout);
0034     varargin{1}.data.setY(varargin{2});
0035     varargout{1} = varargin{1};
0036     return
0037 end
0038
0039
0040 import utils.const.*
0041 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0042
0043 % Collect input variable names
0044 in_names = cell(size(varargin));
0045 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0046
0047 % Collect all AOs
0048 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0049 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0050
0051 %%% If pls contains only one plist with the only key 'name' then set the
0052 %%% property with a plist.
0053 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'y')
0054     rest{1} = find(pls, 'y');
0055 end
0056
0057 if numel(rest) ~= 1
0058     error('### Please specify a value for Y, either in a plist or directly.');
```

```
0098 % Build info object
0099 ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: setY.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106 pl = plist('Y', []);
0107 end
0108
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setYunits

PURPOSE ^

SETYUNITS sets the 'yunits' property of the ao.

SYNOPSIS ^

function varargout = setYunits(varargin)

DESCRIPTION ^

```
SETYUNITS sets the 'yunits' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETYUNITS sets the 'yunits' property of the ao.

CALL:      ao = setYunits(ao, val)
           obj = obj.setYunits(plist('yunits', 'volt'));

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setYunits')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setYunits', 'None')

VERSION:    $Id: setYunits.m,v 1.9 2008/09/05 14:15:33 hewitson Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETYUNITS sets the 'yunits' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETYUNITS sets the 'yunits' property of the ao.
0005 %
0006 % CALL:      ao = setYunits(ao, val)
0007 %           obj = obj.setYunits(plist('yunits', 'volt'));
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('setYunits')
0011 %
```



```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setYunits', 'None')
0014 %
0015 % VERSION:      $Id: setYunits.m,v 1.9 2008/09/05 14:15:33 hewitson Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = setYunits(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032
0033     %%% decide whether we modify the first object, or create a new one.
0034     varargin{1} = copy(varargin{1}, nargout);
0035
0036     varargin{1}.data.setYunits(varargin{2});
0037     varargout{1} = varargin{1};
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all AOs
0049 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051
0052 %%% If pls contains only one plist with the only key 'yunits' then set the
0053 %%% property with a plist.
0054 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls,
'yunits')
0055     rest{1} = find(pls, 'yunits');
0056 end
0057
0058 if numel(rest) ~= 1
0059     error('### Please specify a value for yunits, either in a plist or directly.');
```

```
0097     end
0098     % Build info object
0099     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: setYunits.m,v 1.9
2008/09/05 14:15:33 hewitson Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106     pl = plist('yunits', unit);
0107 end
0108
```

setZ

PURPOSE ^

SETZ sets the 'z' property of the ao.

SYNOPSIS ^

function varargout = setZ(varargin)

DESCRIPTION ^

```
SETZ sets the 'z' property of the ao.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETZ sets the 'z' property of the ao.

CALL:      ao = setZ(ao, val)
           obj = obj.setZ(plist('z', [1 2 3]));

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setZ')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setZ', 'None')

VERSION:    $Id: setZ.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETZ sets the 'z' property of the ao.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETZ sets the 'z' property of the ao.
0005 %
0006 % CALL:      ao = setZ(ao, val)
0007 %           obj = obj.setZ(plist('z', [1 2 3]));
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %           >> ao.getInfo('setZ')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> ao.getInfo('setZ', 'None')
0014 %
0015 % VERSION:      $Id: setZ.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = setZ(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 %%% Internal call: Only one object + don't look for a plist
0031 if strcmp(varargin{end}, 'internal')
0032
0033     %%% decide whether we modify the first object, or create a new one.
0034     varargin{1} = copy(varargin{1}, nargout);
0035
0036     varargin{1}.data.setZ(varargin{2});
0037     varargout{1} = varargin{1};
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all AOs
0049 [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051
0052 %%% If pls contains only one plist with the only key 'z' then set the
0053 %%% property with a plist.
0054 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'z')
0055     rest{1} = find(pls, 'z');
0056 end
0057
0058 if numel(rest) ~= 1
0059     error('### Please specify a value for z, either in a plist or directly.');
```

```
0098 % Build info object
0099 ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: setZ.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106     pl = plist('z', '');
0107 end
0108
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input Analysis object(s).

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input Analysis object(s).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: STRING writes a command string that can be used to recreate the
             input Analysis object(s).

CALL:       cmd = string(a1)

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('string')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('string', 'None')

VERSION:    $Id: string.m,v 1.10 2008/09/05 11:05:29 ingo Exp $

HISTORY:    29-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input Analysis object(s).

This function is called by:

 [string](#) STRING writes a command string that can be used to recreate the input Analysis object(s).

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input Analysis
0002 object(s).
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input Analysis object(s).
0006 %
0007 % CALL:       cmd = string(a1)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %               >> ao.getInfo('string')
0011 %
0012 %               Get information about a specified set-plist by calling:
```

```

0013 %                >> ao.getInfo('string', 'None')
0014 %
0015 % VERSION:      $Id: string.m,v 1.10 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      29-03-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%
0021
0022 function varargout = string(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 import utils.const.*
0031 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs and plists
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039
0040 % Loop over AOs
0041 cmd = '[';
0042 for j=1:length(as)
0043     if isempty(as(j).hist.plistUsed)
0044         error('### this AO was not created with a plist. Can''t convert to string.');
```

```

0045     end
0046     plstr = string(as(j).hist.plistUsed);
0047     cmd = [cmd 'ao(' ' ...
0048         plstr ...
0049         ') '];
0050 end
0051 cmd = [cmd ']''];
0052 if strcmp(cmd, '['')
0053     cmd = '';
0054 end
0055
0056 % Set output
0057 varargout{1} = cmd;
0058 end
0059
0060 %-----
0061 % Get Info Object
0062 %-----
0063 function ii = getInfo(varargin)
0064     if nargin == 1 && strcmpi(varargin{1}, 'None')
0065         sets = {};
0066         pl = [];
0067     else
0068         sets = {'Default'};
0069         pl = getDefaultPlist;
0070     end
0071 % Build info object
0072 ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: string.m,v 1.10
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0073 end
0074
0075 %-----
0076 % Get Default Plist
0077 %-----
0078 function pl_default = getDefaultPlist()
0079     pl_default = plist();
0080 end
0081 % END
0082

```

timeshift

PURPOSE ^

TIMESHIFT for AO/tsdata objects, shifts the time axis such that x(1) = 0.

SYNOPSIS ^

function varargout = timeshift(varargin)

DESCRIPTION ^

```
TIMESHIFT for AO/tsdata objects, shifts the time axis such that x(1) = 0.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: TIMESHIFT for AO/tsdata objects, shifts the time axis such
             that x(1) = 0. The value of t0 is set appropriately.

CALL:       b = timeshift(a)

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('timeshift')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('timeshift', 'None')

VERSION:    $Id: timeshift.m,v 1.11 2008/09/05 11:05:29 ingo Exp $

HISTORY:    23-02-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY makes a (deep) copy of the input AOs.
-  **t0** T0 Get the data property 't0'.

This function is called by:

SUBFUNCTIONS ^

-  **function ii = getInfo(varargin)**
-  **function pl_default = getDefaultPlist()**

SOURCE CODE ^

```
0001 % TIMESHIFT for AO/tsdata objects, shifts the time axis such that x(1) = 0.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TIMESHIFT for AO/tsdata objects, shifts the time axis such
0005 %             that x(1) = 0. The value of t0 is set appropriately.
0006 %
0007 % CALL:       b = timeshift(a)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %             >> ao.getInfo('timeshift')
0011 %
0012 %             Get information about a specified set-plist by calling:
0013 %             >> ao.getInfo('timeshift', 'None')
0014 %
```



```

0015 % VERSION:      $Id: timeshift.m,v 1.11 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:       23-02-08 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = timeshift(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 import utils.const.*
0031 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs and plists
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Check input analysis object
0044 for j=1:numel(bs)
0045     % Which data type do we have
0046     switch class(bs(j).data)
0047     case 'tsdata'
0048         x0 = bs(j).data.getX(1);
0049         t0 = bs(j).data.t0.utc_epoch_milli + 1000*x0;
0050         bs(j).setX(bs(j).data.getX - x0, 'internal');
0051         bs(j).setT0(t0, 'internal');
0052         bs(j).data.collapseX;
0053         % Add history
0054         bs(j).addHistory(getInfo, plist(), ao_invars(j), bs(j).hist);
0055     case {'fsdata', 'cdata', 'xydata'}
0056         error('### I don''t work for frequency-series, xy and constant data.');
```

validate

PURPOSE ^

VALIDATE checks that the input Analysis Object is reproducible and valid.

SYNOPSIS ^

function varargout = validate(varargin)

DESCRIPTION ^

```
VALIDATE checks that the input Analysis Object is reproducible and valid.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: VALIDATE checks that the input Analysis Object is
             reproducible and valid.

CALL:       b = validate(a)

INPUTS:     a - a vector, matrix or cell array of Analysis Objects

OUTPUTS:    b - a vector, matrix or cell array of logical results:
             0 - input object failed
             1 - input object passed

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('validate')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('validate', 'None')

VERSION:    $Id: validate.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    22-02-08 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [sum](#) SUM computes the sum of the data in the AO.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % VALIDATE checks that the input Analysis Object is reproducible and valid.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: VALIDATE checks that the input Analysis Object is
0005 %               reproducible and valid.
0006 %
0007 % CALL:       b = validate(a)
0008 %
```

```

0009 % INPUTS:      a - a vector, matrix or cell array of Analysis Objects
0010 %
0011 % OUTPUTS:     b - a vector, matrix or cell array of logical results:
0012 %              0 - input object failed
0013 %              1 - input object passed
0014 %
0015 % M-FILE INFO:  Get information about this methods by calling
0016 %              >> ao.getInfo('validate')
0017 %
0018 %              Get information about a specified set-plist by calling:
0019 %              >> ao.getInfo('validate', 'None')
0020 %
0021 % VERSION:      $Id: validate.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0022 %
0023 % HISTORY:      22-02-08 M Hewitson
0024 %              Creation
0025 %
0026 %%%%%%%%%%%
0027
0028 function varargout = validate(varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 import utils.const.*
0037 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0038
0039 % Collect input variable names
0040 in_names = cell(size(varargin));
0041 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0042
0043 % Collect all AOs and plists
0044 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0045
0046 % --- Initial set up
0047 versions = []; % initialised with function: ltpda_versions
0048 ltpda_versions;
0049 passed = zeros(1, numel(as));
0050
0051 % Check each input analysis object
0052 for ec=1:numel(as)
0053     % get a list of files that built this object
0054     [n,a,nodes] = getNodes(as(ec).hist, [] ,0,1,[]);
0055
0056     % Initialise the match vector
0057     matches = zeros(length(nodes),1);
0058
0059     % Loop over each history node, i.e., each function
0060     for jj=1:length(nodes)
0061         % This node
0062         node = nodes(jj);
0063         % Assume failure to start with
0064         matches(jj) = 0;
0065         % get fcn name
0066         fcname = node.names;
0067         % Find all functions in MATLAB path with this name
0068         mfiles = which(fcname, '-ALL');
0069         % Find all matches in versions{}
0070         idx = ismember(versions(:,2), fcname);
0071         ltpdafiles = versions(find(idx==1), 1);
0072         % check against each one found
0073         for kk=1:length(mfiles)
0074             mfile = mfiles{kk};
0075             % make file hash
0076             try
0077                 % Load the full file contents
0078                 fd = fopen(mfile, 'r');
0079                 fc = fscanf(fd, '%s');
0080                 fclose(fd);
0081                 % Make MD5 hash
0082                 mhash = utils.prog.hash(fc, 'MD5');
0083                 % Check against all ltpda files with this function name
0084                 for ll=1:length(ltpdafiles)
0085                     % this file
0086                     lfile = ltpdafiles{ll};
0087                     % Load file contents
0088                     fd = fopen(lfile, 'r');
0089                     fc = fscanf(fd, '%s');
0090                     fclose(fd);
0091                     % Make MD5 hash
0092                     lhash = utils.prog.hash(fc, 'MD5');
0093                     % Compares hashes
0094                     if strcmp(mhash, lhash)

```

```

0095         matches(jj) = 1;
0096     end
0097 end
0098 catch
0099     warning('!!! failed to test against: %s', mfile);
0100 end
0101 end % End loop over files on MATLAB path
0102
0103 if matches(jj)==0
0104     fails = which(fcnname, '-ALL');
0105     for ff=1:length(fails)
0106         utils.helper.msg(msg.PROC1, 'Illegal function: %s', fails{ff});
0107     end
0108 end
0109 end % end loop over nodes
0110
0111 % Decide whether or not this AO is valid
0112 if sum(matches) == length(nodes)
0113     passed(ec) = 1;
0114     utils.helper.msg(msg.PROC1, 'AO validated');
0115 else
0116     passed(ec) = 0;
0117     utils.helper.msg(msg.PROC1, 'AO not validated');
0118 end
0119 end % end loop over all objects
0120
0121 % Set outputs
0122 varargout{1} = passed;
0123 end
0124
0125 %-----
0126 % Get Info Object
0127 %-----
0128 function ii = getInfo(varargin)
0129     if nargin == 1 && strcmpi(varargin{1}, 'None')
0130         sets = {};
0131         pl = [];
0132     else
0133         sets = {'Default'};
0134         pl = getDefaultPlist;
0135     end
0136     % Build info object
0137     ii = minfo(mfilename, 'ao', '', utils.const.categories.helper, '$Id: validate.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0138 end
0139
0140 %-----
0141 % Get Default Plist
0142 %-----
0143 function pl_default = getDefaultPlist()
0144     pl_default = plist();
0145 end
0146 % END
0147

```

isprop

PURPOSE ^

ISPROP tests if the given field is one of the object properties.

SYNOPSIS ^

function varargout = isprop(varargin)

DESCRIPTION ^

ISPROP tests if the given field is one of the object properties.
%%%

DESCRIPTION: ISPROP tests if the given field is one of the object properties.

CALL: obj.isprop('field');
 isprop(obj, 'field');

INPUTS: obj - Input objects
 field - Property name of the object

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('isprop')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('isprop', 'set')

VERSION: \$Id: isprop.m,v 1.2 2008/09/04 15:29:30 ingo Exp \$

HISTORY: 25-09-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_obj](#) LTPDA_OBJ is the abstract ltpda base class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ISPROP tests if the given field is one of the object properties.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ISPROP tests if the given field is one of the object properties.
0005 %
0006 % CALL:           obj.isprop('field');
0007 %                   isprop(obj, 'field');
0008 %
0009 % INPUTS:         obj     - Input objects
0010 %                   field  - Property name of the object
0011 %
0012 % M-FILE INFO:    Get information about this methods by calling
0013 %                   >> ao.getInfo('isprop')
```

```

0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> ao.getInfo('isprop', 'set')
0017 %
0018 % VERSION:      $Id: isprop.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0019 %
0020 % HISTORY: 25-09-07 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024 %
0025 function varargout = isprop(varargin)
0026
0027     %% Check if this is a call for parameters
0028     if utils.helper.isinfocall(varargin{:})
0029         varargout{1} = getInfo(varargin{3});
0030         return
0031     end
0032
0033     %% Collect input variable names
0034     in_names = cell(size(varargin));
0035     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037     %% Collect all objects
0038     [objs, invars, rest] = utils.helper.collect_objects(varargin(:), '', in_names);
0039
0040     %% If we eliminated the objects and plists then is the rest the property name
0041     if length(rest) == 1
0042         field = rest{1};
0043     else
0044         error('### Please specify [only one] field-name.')
0045     end
0046
0047     res = zeros(size(objs));
0048     try
0049         dummy = objs(1).(field);
0050         res(:) = 1;
0051         varargout{ii} = res;
0052     catch
0053         res(:) = 0;
0054     end
0055
0056     %% prepare output
0057     varargout{1} = res;
0058 end
0059
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061 %                               Local Functions                               %
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065 %
0066 % FUNCTION:      getInfo
0067 %
0068 % DESCRIPTION: Get Info Object
0069 %
0070 % HISTORY:      11-07-07 M Hewitson
0071 %               Creation.
0072 %
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074
0075 function ii = getInfo(varargin)
0076     if nargin == 1 && strcmpi(varargin{1}, 'None')
0077         sets = {};
0078         pl = [];
0079     else
0080         sets = {'Default'};
0081         pl = getDefaultPlist;
0082     end
0083     % Build info object
0084     ii = minfo(mfilename, 'ltpda_obj', '', utils.const.categories.helper, '$Id: isprop.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0085 end
0086
0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088 %
0089 % FUNCTION:      getDefaultPlist
0090 %
0091 % DESCRIPTION: Get Default Plist
0092 %
0093 % HISTORY:      11-07-07 M Hewitson
0094 %               Creation.
0095 %
0096 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0097
0098 function plo = getDefaultPlist()

```

```
0099   plo = plist();  
0100 end  
0101
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setName

PURPOSE ^

SETNAME Set the property 'name'.

SYNOPSIS ^

function varargout = setName(varargin)

DESCRIPTION ^

```
SETNAME Set the property 'name'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'name'.

CALL:      obj = obj.setName('new name');
           obj = obj.setName(plist('name', 'new name'));
           obj = setName(obj, 'new name');

EXAMPLE:   obj.setName -> Sets the name to the variable name.
           In this case to 'obj'

INPUTS:    obj - can be a vector, matrix, list, or a mix of them.
           pl  - to set the name with a plist specify only one plist with
           only one key-word 'name'.

M-FILE INFO: Get information about this methods by calling
           >> ao.getInfo('setName')

           Get information about a specified set-plist by calling:
           >> ao.getInfo('setName', 'set')

VERSION:   $Id: setName.m,v 1.4 2008/09/04 15:29:30 ingo Exp $

HISTORY:   27-05-2008 Diepholz
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETNAME Set the property 'name'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'name'.
0005 %
0006 % CALL:      obj = obj.setName('new name');
0007 %           obj = obj.setName(plist('name', 'new name'));
0008 %           obj = setName(obj, 'new name');
```



```

0009 %
0010 % EXAMPLE:      obj.setName -> Sets the name to the variable name.
0011 %              In this case to 'obj'
0012 %
0013 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0014 %              pl  - to set the name with a plist specify only one plist with
0015 %                    only one key-word 'name'.
0016 %
0017 % M-FILE INFO:  Get information about this methods by calling
0018 %              >> ao.getInfo('setName')
0019 %
0020 %              Get information about a specified set-plist by calling:
0021 %              >> ao.getInfo('setName', 'set')
0022 %
0023 % VERSION:      $Id: setName.m,v 1.4 2008/09/04 15:29:30 ingo Exp $
0024 %
0025 % HISTORY:      27-05-2008 Diepholz
0026 %              Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 function varargout = setName(varargin)
0031
0032 %%% Check if this is a call for parameters
0033 if utils.helper.isinfocall(varargin{:})
0034     varargout{1} = getInfo(varargin{3});
0035     return
0036 end
0037
0038 %%% Internal call: Only one object + don't look for a plist
0039 if strcmp(varargin{end}, 'internal')
0040
0041     %%% decide whether we modify the first object, or create a new one.
0042     varargin{1} = copy(varargin{1}, nargout);
0043
0044     varargin{1}.name = varargin{2};
0045     varargout{1} = varargin{1};
0046     return
0047 end
0048
0049 %%% Collect input variable names
0050 in_names = cell(size(varargin));
0051 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0052
0053 [objs, obj_invars, rest] = utils.helper.collect_objects(varargin(:), '', in_names);
0054 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0055
0056 %%% If pls contains only one plist with the only key 'name' then set the
0057 %%% property with a plist.
0058 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'name')
0059     rest{1} = find(pls, 'name');
0060 end
0061
0062 if numel(rest) > 1
0063     error('### Please specify a single name, either in a plist or directly.');
```

```
0095
0096
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098 %                                     Local Functions                                     %
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100
0101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0102 %
0103 % FUNCTION:      getInfo
0104 %
0105 % DESCRIPTION:   Get Info Object
0106 %
0107 % HISTORY:      11-07-07 M Hewitson
0108 %                Creation.
0109 %
0110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111
0112 function ii = getInfo(varargin)
0113     if nargin == 1 && strcmpi(varargin{1}, 'None')
0114         sets = {};
0115         pl   = [];
0116     else
0117         sets = {'Default'};
0118         pl   = getDefaultPlist;
0119     end
0120     % Build info object
0121     ii = minfo(mfilename, 'ltpda_uoh', '', utils.const.categories.helper, '$Id:
setName.m,v 1.4 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0122 end
0123
0124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0125 %
0126 % FUNCTION:      getDefaultPlist
0127 %
0128 % DESCRIPTION:   Get Default Plist
0129 %
0130 % HISTORY:      11-07-07 M Hewitson
0131 %                Creation.
0132 %
0133 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0134
0135 function plo = getDefaultPlist()
0136     plo = plist('name', '');
0137 end
0138
```

addHistory

PURPOSE ^

ADDHISTORY Add a history-object to the ltpda_uo object.

SYNOPSIS ^

function varargout = addHistory(varargin)

DESCRIPTION ^

```
ADDHISTORY Add a history-object to the ltpda_uo object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Add a history-object to the ltpda_uoh object.

CALL:      obj = addHistory(obj, minfo, h_pl, var_name, inhists, ...);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('addHistory')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('addHistory', 'None')

VERSION:    $Id: addHistory.m,v 1.7 2008/09/04 15:29:30 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ADDHISTORY Add a history-object to the ltpda_uo object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Add a history-object to the ltpda_uoh object.
0005 %
0006 % CALL:      obj = addHistory(obj, minfo, h_pl, var_name, inhists, ...);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> ao.getInfo('addHistory')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> ao.getInfo('addHistory', 'None')
0013 %
0014 % VERSION:    $Id: addHistory.m,v 1.7 2008/09/04 15:29:30 ingo Exp $
0015 %
0016 % HISTORY:    27-05-2008 Diepholz
0017 %              Creation
```

```

0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = addHistory(varargin)
0022
0023     %% Check if this is a call for parameters
0024     if utils.helper.isinfocall(varargin{:})
0025         varargout{1} = getInfo(varargin{3});
0026         return
0027     end
0028
0029     % The object to add history to
0030     obj = varargin{1};
0031
0032     %% Decide on a deep copy or a modify
0033     obj = copy(obj, nargout);
0034
0035     %% Add history to all objects
0036     for ii = 1:numel(obj)
0037         obj(ii).setHist(history(time().utc_epoch_milli, varargin{2}, varargin{3},
0038         varargin{4}, varargin{5:end}));
0039     end
0040
0041     %% Prepare output
0042     varargout{1} = obj;
0043 end
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0046 % Local Functions
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:       11-07-07 M Hewitson
0055 %                 Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'ltpda_uoh', '', utils.const.categories.internal, '$Id:
0069 addHistory.m,v 1.7 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0070 end
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:       11-07-07 M Hewitson
0078 %                 Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081
0082 function plo = getDefaultPlist()
0083     plo = plist();
0084 end
0085

```

mdc1_ifo2acc_fd

PURPOSE ^

MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.

SYNOPSIS ^

function varargout = mdc1_ifo2acc_fd(varargin)

DESCRIPTION ^

```
MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: MDC1_IFO2ACC_FS calculates the external acceleration in
              the frequency-domain for MDC1.

It computes: a = [DS^-1 + C]o

CALL:        b = mdc1_ifo2acc_fd(o1xx,o12xx,pl)

INPUTS:      o1xx      - spectral estimate of the IFO output o12
                  [default: empty ao]
              o12xx     - spectral estimate of the IFO output o12
                  [default: empty ao]
              o112xx    - cross-spectral estimate of the IFO output o1 and o12
                  [default: empty ao]

PARAMETERS:

              'Omega1'  - the square of the stiffness term for the dynamical response
                          of test-mass 1 coupling to SC [default: 1.3e-6]
              'Omega3'  - the square of the stiffness term for the dynamical response
                          of test-mass 2 coupling to SC [default: 2e-6]
              'delta'   - the cross-coupling factor of o1 into o12 [default: -1e-4]

VERSION:      $Id: mdc1_ifo2acc_fd.m,v 1.4 2008/09/05 11:05:29 ingo Exp $

M-FILE INFO:  Get information about this methods by calling
              >> ao.getInfo('mdc1_ifo2acc_fd')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mdc1_ifo2acc_fd', 'None')

HISTORY: 11-04-08 M Hewitson
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [ao](#) AO analysis object class constructor.
-  [conj](#) CONJ overloads the conjugate operator for Analysis objects.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [sqrt](#) SQRT computes the square root of the data in the AO.

This function is called by:

SUBFUNCTIONS ^

 function [a11xx, a22xx] = computeAcc(f, w1, w3, delta, o1xx, o12xx, o112xx)
 function ii = getInfo(varargin)
 function plo = getDefaultPlist()

SOURCE CODE ^

```

0001 % MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: MDC1_IFO2ACC_FS calculates the external acceleration in
0006 %               the frequency-domain for MDC1.
0007 %
0008 % It computes: a = [DS^-1 + C]o
0009 %
0010 % CALL:         b = mdc1_ifo2acc_fd(o1xx,o12xx,pl)
0011 %
0012 % INPUTS:       o1xx      - spectral estimate of the IFO output o12
0013 %               [default: empty ao]
0014 %               o12xx     - spectral estimate of the IFO output o12
0015 %               [default: empty ao]
0016 %               o112xx    - cross-spectral estimate of the IFO output o1 and o12
0017 %               [default: empty ao]
0018 % PARAMETERS:
0019 %
0020 %               'Omega1'  - the square of the stiffness term for the dynamical response
0021 %                           of test-mass 1 coupling to SC [default: 1.3e-6]
0022 %               'Omega3'  - the square of the stiffness term for the dynamical response
0023 %                           of test-mass 2 coupling to SC [default: 2e-6]
0024 %               'delta'   - the cross-coupling factor of o1 into o12 [default: -1e-4]
0025 %
0026 % VERSION:      $Id: mdc1_ifo2acc_fd.m,v 1.4 2008/09/05 11:05:29 ingo Exp $
0027 %
0028 % M-FILE INFO:  Get information about this methods by calling
0029 %               >> ao.getInfo('mdc1_ifo2acc_fd')
0030 %
0031 %               Get information about a specified set-plist by calling:
0032 %               >> ao.getInfo('mdc1_ifo2acc_fd', 'None')
0033 %
0034 % HISTORY: 11-04-08 M Hewitson
0035 %           Creation
0036 %
0037 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0038 function varargout = mdc1_ifo2acc_fd(varargin)
0039
0040 % Check if this is a call for parameters
0041 if utils.helper.isinfocall(varargin{:})
0042     varargout{1} = getInfo(varargin{3});
0043     return
0044 end
0045
0046 import utils.const.*
0047 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0048
0049 % Collect input variable names
0050 in_names = cell(size(varargin));
0051 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0052
0053 % Collect all AOs and plists
0054 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0055 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0056 % Decide on a deep copy or a modify
0057 bs = copy(as, nargout);
0058
0059 % combine plists
0060 pl = combine(pl, getDefaultPlist());
0061 %% Extract parameters from plist
0062
0063 % Get parameters out
0064 w1 = find(pl, 'Omega1');
0065 w3 = find(pl, 'Omega3');
0066 delta = find(pl, 'delta');
0067 o1xx = bs(1);
0068 o12xx = bs(2);
0069 o112xx = bs(3);
0070 if ~strcmp(o1xx.name, 'o1xx')
0071     warning('!!! The name of the first input AO is no ''o1''. Are you sure this is the
o1xx frequency-series?');
0072 end
0073 if ~strcmp(o12xx.name, 'o12xx')
0074     warning('!!! The name of the second input AO is no ''o12''. Are you sure this is

```

```

the o12xx frequency-series?');
0075     end
0076
0077     if isempty(o1xx.data) || isempty(o12xx.data) || isempty(o112xx.data)
0078         error('### Please provide two input frequency-series.');
```

```

0079     end
0080     if ~isequal(o1.data.x, o12.data.x, o112xx.data.x)
0081         error('### The three input frequency-series should be computed at the same
frequencies.');
```

```

0082     end
0083
0084     % Get frequency vector from one of the input spectra
0085     f = o1xx.data.getX;
0086
0087     %% Compute response for frequencies f
0088
0089     [a11xx, a22xx] = computeAcc(f, w1, w3, delta, o1xx, o12xx, o112xx);
0090
0091     varargout{1} = a11xx;
0092     varargout{2} = a22xx;
0093 end
0094 %-----
0095 % Get DF controller for each frequency
0096 function [a11xx, a22xx] = computeAcc(f, w1, w3, delta, o1xx, o12xx, o112xx)
0097
0098     % A model of the downsampling from 100 to 10Hz
0099     pl = plist('gain',1, 'poles', [pz(10), pz(10), pz(10)], 'zeros', []);
0100     pzm = pzmodel(pl);
0101     ds = resp(pzm, plist('f', f));
0102
0103     % Drag-free
0104     Cdf = ltpda_mdc1_C(plist('f', f, 'Controller', 'df'));
0105     Adf = ltpda_mdc1_actuator(plist('f', f, 'Actuator', 'df'));
0106     Cdf = ds.*Cdf.*Adf;
0107     Sw1 = ltpda_mdc1_dynamics(plist('f', f, 'Omega2', w1.data.y.^2));
0108     Sw1.setName('Sw1', 'internal');
```

```

0109
0110     % Suspension
0111     Csus = ltpda_mdc1_C(plist('f', f, 'Controller', 'sus'));
0112     Asus = ltpda_mdc1_actuator(plist('f', f, 'Actuator', 'sus'));
0113     Csus = ds.*Csus.*Asus;
0114     Sw3 = ltpda_mdc1_dynamics(plist('f', f, 'Omega2', w3.data.y.^2));
0115     Sw3.setName('Sw3', 'internal');
```

```

0116
0117     % Square terms for dealing with PSDs
0118     Csusxx = abs(Csus).^2;
0119     Cdfxx = abs(Cdf).^2;
0120     Sw1xx = abs(Sw1).^2;
0121     Sw3xx = abs(Sw3).^2;
0122
0123     wd = w3.^2 - w1.^2;
0124     beta = wd - delta.*Sw3;
0125     bxx = abs(beta).^2;
0126
0127     % Calibrate back to a1
0128
0129     a11xx = o1xx .* (Sw1xx + Cdfxx - Cdf.*conj(Sw1) - conj(Cdf).*Sw1);
0130     a11xx.setName('PSD(a11)', 'internal');
```

```

0131
0132     % Calibrate back to a2
0133
0134     e1 = o1xx.*bxx;
0135     e2 = sqrt(o12xx).*Sw3;
0136     e3 = sqrt(o12xx).*Csus;
0137     a22t1 = e1 + abs(e2 - e3).^2;
0138
0139     a22t2 = beta.*(conj(Sw3)+conj(Csus)).*o112xx;
0140     a22t3 = conj(beta).*(Sw3+Csus).*o112xx;
0141     a22xx = abs(a22t1 ...
0142         + a22t2 ...
0143         + a22t3);
0144     a22xx.setName('PSD(a22)', 'internal');
```

```

0145 end
0146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0147 %                                     Local Functions
0148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0149
0150 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0151 %
0152 % FUNCTION:      getInfo
0153 %
0154 % DESCRIPTION:  Get Info Object
0155 %
0156 % HISTORY:      11-07-07 M Hewitson
0157 %               Creation.
0158 %

```



```
0159 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0160
0161 function ii = getInfo(varargin)
0162     if nargin == 1 && strcmpi(varargin{1}, 'None')
0163         sets = {};
0164         pl = [];
0165     else
0166         sets = {'Default'};
0167         pl = getDefaultPlist;
0168     end
0169     % Build info object
0170     ii = minfo(mfilename, 'ao', '', utils.const.categories.mdc01, '$Id:
mdc1_ifo2acc_fd.m,v 1.4 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0171 end
0172
0173 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0174 %
0175 % FUNCTION:      getDefaultPlist
0176 %
0177 % DESCRIPTION:  Get Default Plist
0178 %
0179 % HISTORY:      11-07-07 M Hewitson
0180 %               Creation.
0181 %
0182 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0183
0184 function plo = getDefaultPlist()
0185     plo = plist('Omega1', 1.3e-6, ...
0186         'Omega3', 2e-6, ...
0187         'delta', -1e-4, ...
0188         'o1xx', ao, ...
0189         'o12xx', ao, ...
0190         'o112xx', ao);
0191 end
0192
0193
```


mdc1_ifo2acc_inloop

PURPOSE ^

MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.

SYNOPSIS ^

function varargout = mdc1_ifo2acc_inloop(varargin)

DESCRIPTION ^

```
MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MDC1_IFO2ACC_INLOOP calculates the inloop acceleration
             in the time-domain for MDC1.

It computes: a = DS^-1 o

CALL:      [a1,a2] = mdc1_ifo2acc_inloop(o1,o12,pl)

INPUTS:    o1      - time-series of the IFO output o1
            o12     - time-series of the IFO output o12

PARAMETERS: 'Omega1' - the square of the stiffness term for the dynamical response
              of test-mass 1 coupling to SC [default: 1.3e-6]
            'Omega3' - the square of the stiffness term for the dynamical response
              of test-mass 2 coupling to SC [default: 2e-6]
            'delta'  - the cross-coupling factor of o1 into o12 [default: -1e-4]

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('mdc1_ifo2acc_inloop')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mdc1_ifo2acc_inloop', 'None')

VERSION:    $Id: mdc1_ifo2acc_inloop.m,v 1.10 2008/09/05 14:15:14 hewitson Exp $

HISTORY:    11-04-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [mdc1_x2acc](#) MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter

This function is called by:

SUBFUNCTIONS ^

-  [function \[a1, a2\] = computeAcc\(w1, w3, delta, o1, o12, ao_invars\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MDC1_IFO2ACC_INLOOP calculates the inloop acceleration
0005 %               in the time-domain for MDC1.
0006 %
0007 % It computes:  $a = DS^{-1} o$ 
0008 %
0009 % CALL:         [a1,a2] = mdc1_ifo2acc_inloop(o1,o12,pl)
0010 %
0011 % INPUTS:       o1           - time-series of the IFO output o1
0012 %               o12          - time-series of the IFO output o12
0013 %
0014 % PARAMETERS:   'Omega1'     - the square of the stiffness term for the dynamical response
0015 %               of test-mass 1 coupling to SC [default: 1.3e-6]
0016 %               'Omega3'     - the square of the stiffness term for the dynamical response
0017 %               of test-mass 2 coupling to SC [default: 2e-6]
0018 %               'delta'      - the cross-coupling factor of o1 into o12 [default: -1e-4]
0019 %
0020 % M-FILE INFO:  Get information about this methods by calling
0021 %               >> ao.getInfo('mdc1_ifo2acc_inloop')
0022 %
0023 %               Get information about a specified set-plist by calling:
0024 %               >> ao.getInfo('mdc1_ifo2acc_inloop', 'None')
0025 %
0026 % VERSION:      $Id: mdc1_ifo2acc_inloop.m,v 1.10 2008/09/05 14:15:14 hewitson Exp $
0027 %
0028 % HISTORY:      11-04-08 M Hewitson
0029 %               Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 function varargout = mdc1_ifo2acc_inloop(varargin)
0034 %
0035 % Check if this is a call for parameters
0036 if utils.helper.isinfocall(varargin{:})
0037     varargout{1} = getInfo(varargin{3});
0038     return
0039 end
0040 %
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043 %
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047 %
0048 % Collect all AOs and plists
0049 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0051 %
0052 % Decide on a deep copy or a modify
0053 bs = copy(as, nargout);
0054 %
0055 % combine plists
0056 pl = combine(pl, getDefaultPlist());
0057 %
0058 %% Extract parameters from plist
0059 %
0060 % Get parameters out
0061 w1 = find(pl, 'Omega1');
0062 w3 = find(pl, 'Omega3');
0063 delta = find(pl, 'delta');
0064 o1 = bs(1);
0065 o12 = bs(2);
0066 %
0067 if ~strcmp(o1.name, 'o1')
0068     warning('!!! The name of the first input AO is not ''o1''. Are you sure this is the
0069 o1 time-series?');
0070 end
0071 if ~strcmp(o12.name, 'o12')
0072     warning('!!! The name of the second input AO is not ''o12''. Are you sure this is
0073 the o1 time-series?');
0074 end
0075 if isempty(o1.data) || isempty(o12.data)
0076     error('### Please provide two input time-series.');
```

```

0085
0086     a1.setYunits('m s^-2');
0087     a2.setYunits('m s^-2');
0088
0089     if nargout == 2
0090         varargout{1} = a1;
0091         varargout{2} = a2;
0092     elseif nargout == 1
0093         varargout{1} = [a1 a2];
0094     end
0095 end
0096
0097 %-----
0098 % Get DF controller for each frequency
0099 function [a1, a2] = computeAcc(w1, w3, delta, o1, o12, ao_invars)
0100
0101     if isa(w1, 'ao')
0102         w1 = w1.data.y;
0103     end
0104     if isa(w3, 'ao')
0105         w3 = w3.data.y;
0106     end
0107
0108     %---- compute a1
0109     pl = plist('Omega2', w1);
0110     a1 = mdc1_x2acc(o1, pl);
0111     % Add history
0112     a1.addHistory(getInfo, pl, ao_invars, [o1.hist o12.hist]);
0113     % Set AO name
0114     a1.setName(sprintf('%s(%s,%s)', mfilename, ao_invars{1}, ao_invars{2}), 'internal');
0115
0116     % compute a2
0117     pl = plist('Omega2', w3);
0118     wd = w3-w1; % omega_delta
0119     a2 = wd.*o1 - delta.*mdc1_x2acc(o1, pl) + mdc1_x2acc(o12, pl);
0120     % Add history
0121     a2.addHistory(getInfo, pl, ao_invars, [o1.hist o12.hist]);
0122     % Set AO name
0123     a2.setName(sprintf('%s(%s,%s)', mfilename, ao_invars{1}, ao_invars{2}), 'internal');
0124 end
0125
0126 %-----
0127 % Get Info Object
0128 %-----
0129 function ii = getInfo(varargin)
0130     if nargin == 1 && strcmpi(varargin{1}, 'None')
0131         sets = {};
0132         pl = [];
0133     else
0134         sets = {'Default'};
0135         pl = getDefaultPlist;
0136     end
0137     % Build info object
0138     ii = minfo(mfilename, 'ao', '', utils.const.categories.mdc01, '$Id:
mdc1_ifo2acc_inloop.m,v 1.10 2008/09/05 14:15:14 hewitson Exp $', sets, pl);
0139 end
0140
0141 %-----
0142 % Get Default Plist
0143 %-----
0144 function pl_default = getDefaultPlist()
0145     pl_default = plist(...
0146         'Omega1', 1.3e-6, ...
0147         'Omega3', 2e-6, ...
0148         'delta', -1e-4);
0149 end
0150

```

mdc1_ifo2control

PURPOSE ^

MDC1_IFO2CONTROL converts the input time-series to control forces.

SYNOPSIS ^

function varargout = mdc1_ifo2control(varargin)

DESCRIPTION ^

```
MDC1_IFO2CONTROL converts the input time-series to control forces.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MDC1_IFO2CONTROL converts the input time-series to control forces.

CALL:      c = mdc1_ifo2control(o1, o12)

INPUTS:    o1      - time-series of the IFO output o1
           o12     - time-series of the IFO output o12

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('mdc1_ifo2control')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mdc1_ifo2control', 'None')

VERSION:    $Id: mdc1_ifo2control.m,v 1.8 2008/09/05 14:15:14 hewitson Exp $

HISTORY:    11-04-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function o1 = applyCdf\(o1\)](#)
-  [function o12 = applyCsus\(o12\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MDC1_IFO2CONTROL converts the input time-series to control forces.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MDC1_IFO2CONTROL converts the input time-series to control forces.
0005 %
0006 % CALL:      c = mdc1_ifo2control(o1, o12)
0007 %
0008 % INPUTS:    o1      - time-series of the IFO output o1
0009 %           o12     - time-series of the IFO output o12
0010 %
```

```

0011 % M-FILE INFO: Get information about this methods by calling
0012 % >> ao.getInfo('mdc1_ifo2control')
0013 %
0014 % Get information about a specified set-plist by calling:
0015 % >> ao.getInfo('mdc1_ifo2control', 'None')
0016 %
0017 % VERSION: $Id: mdc1_ifo2control.m,v 1.8 2008/09/05 14:15:14 hewitson Exp $
0018 %
0019 % HISTORY: 11-04-08 M Hewitson
0020 % Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = mdc1_ifo2control(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 import utils.const.*
0033 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs and plists
0040 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0041 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargin);
0045
0046 % combine plists
0047 pl = combine(pl, getDefaultPlist());
0048
0049 %% Extract parameters from plist
0050
0051 % Get parameters out
0052 o1 = bs(1);
0053 o12 = bs(2);
0054
0055 % Store input histories
0056 h1 = o1.hist;
0057 h12 = o12.hist;
0058
0059 % Apply controller Cdf
0060 c1 = applyCdf(o1);
0061 % Add history
0062 c1.addHistory(getInfo, pl, ao_invars, [h1 h12]);
0063 % Set AO name
0064 c1.setName(sprintf('%s(%s,%s)', mfilename, ao_invars{1}, ao_invars{2}), 'internal');
0065
0066 c1.setYunits('m s^-2');
0067
0068 % Apply controller Csus
0069 c2 = applyCsus(o12);
0070 % Add history
0071 c2.addHistory(getInfo, pl, ao_invars, [h1 h12]);
0072 % Set AO name
0073 c2.setName(sprintf('%s(%s,%s)', mfilename, ao_invars{1}, ao_invars{2}), 'internal');
0074
0075 c2.setYunits('m s^-2');
0076
0077 % Set output
0078 if nargin == 1
0079     varargout{1} = [c1 c2];
0080 else
0081     varargout{1} = c1;
0082     varargout{2} = c2;
0083 end
0084 end
0085 %-----
0086 % Apply drag-free controller with filters designed in LISO fit to
0087 %
0088 % pole 0.0000108323f ### fitted (name = pole0)
0089 % pole 492.4469487554m ### fitted (name = pole3)
0090 % zero 5.9923220808m 520.0769724208m ### fitted (name = zero0)
0091 % factor 190214348700.9898376465G ### fitted
0092 %
0093 % pole 88.3387108214u 501.9601697563m ### fitted (name = pole1)
0094 % zero 13.4574095925m 715.2446797808m ### fitted (name = zero1)
0095 % factor 1
0096 %

```

```

0097 % pole 300.6237575544m 968.3134470825m ### fitted (name = pole2)
0098 % zero 5.8794500998 ### fitted (name = zero2)
0099 % zero 40.1685309724 ### fitted (name = zero3)
0100 % factor 1
0101
0102 function o1 = applyCdf(o1)
0103
0104 % Sample frequency filters are designed for
0105 filtfs = 100;
0106
0107 if o1.data.fs ~= filtfs
0108     warning('### Filters are designed for %dHz. Resampling time-series.', filtfs);
0109     fsin = o1.data.fs;
0110     o1.resample(plist('fsout', filtfs));
0111 end
0112
0113 %---- First filter stage
0114 a = [27801.38587094533 -107218.527103749 180226.5071750752 ...
0115      -173891.2206875895 101664.0906613848 -33271.76258436936 4689.527541600029];
0116 b = [1 -3.833870654386819 6.42742334310825 -6.197734830947992 ...
0117      3.622482927835305 -1.185355897309293 0.1670551117005498];
0118 ff1 = miir(a,b,100);
0119
0120 %---- Second filter stage
0121 a = [4.312632196506008e-05 -0.0001281084965932162 0.0001404024385649956 ...
0122      -6.89671676763067e-05 1.354691430208644e-05];
0123 b = [1 -2.971420266964361 3.257128504882159 -1.599992417674869 0.3142841797676335];
0124 ff2 = miir(a,b,100);
0125
0126 %---- Third filter stage
0127 a = [0.00095273516542127803953 -0.00091952972321343779717 5.5197001381098205063e-05
8.7265653018697098768e-05];
0128 b = [1 -2.4840746702182161521 1.9782850928995776574 -0.49403473523860202965];
0129 ff3 = miir(a,b,100);
0130
0131 % Apply filters
0132 o1.filter(ff1);
0133 o1.filter(ff2);
0134 o1.filter(ff3);
0135
0136 % Apply delay filter
0137 o1.delay(plist('N', 31));
0138
0139 % Apply actuator
0140 pzm = pzmodel(1, pz(0.8), []);
0141 af = miir(pzm, plist('fs', filtfs));
0142 o1.filter(af);
0143
0144 % Resample if necessary
0145 if fsin ~= filtfs
0146     o1.resample(plist('fsout', fsin));
0147 end
0148 end
0149
0150 %-----
0151 % Apply suspension controller designed in LISO
0152 %
0153 % pole 13.5111750971m 765.5054182119m ### fitted (name = pole1)
0154 % zero 1.1290293787m 906.1738368029m ### fitted (name = zero0)
0155 % factor 1 ### fitted
0156 %
0157 % pole 0.0000099821f ### fitted (name = pole0)
0158 % pole 17.3189756179m ### fitted (name = pole2)
0159 % zero 10.9038956793 ### fitted (name = zero1)
0160 % delay 27.2832312829m ### fitted
0161 % factor 17126.3998612698e9
0162 %
0163 function o12 = applyCsus(o12)
0164
0165 % Sample frequency filters are designed for
0166 filtfs = 100;
0167
0168 if o12.data.fs ~= filtfs
0169     warning('### Filters are designed for %dHz. Resampling time-series.', filtfs);
0170     fsin = o12.data.fs;
0171     o12.resample(plist('fsout', filtfs));
0172 end
0173
0174 %---- First filter stage
0175 a = [142.9896507914776862 -285.96810764145169514 142.97845756946250617];
0176 b = [1 -1.9988914209045689496 0.99889214047646490702];
0177 ff1 = miir(a,b,100);
0178
0179 %---- Second filter stage
0180 a = [2.146238155432156e-11 -2.556622712681001e-11 1.278311356340501e-11 -
2.840691902978891e-12];

```

```

0181     b = [1 -2.499139379998971 1.998822309472276 -0.4996829294733051];
0182     ff2 = miir(a,b,100);
0183
0184     % Apply filters
0185     o12.filter(ff1);
0186     o12.filter(ff2);
0187
0188     % Apply delay filter
0189     o12.delay(plist('N', 30));
0190
0191     % Apply actuator
0192     pzm = pzmodel(1, pz(8), []);
0193     af = miir(pzm, plist('fs', filtfs));
0194     o12.filter(af);
0195
0196     % Resample if necessary
0197     if fsin ~= filtfs
0198         o12.resample(plist('fsout', fsin));
0199     end
0200 end
0201
0202 %-----
0203 % Get Info Object
0204 %-----
0205 function ii = getInfo(varargin)
0206     if nargin == 1 && strcmpi(varargin{1}, 'None')
0207         sets = {};
0208         pl = [];
0209     else
0210         sets = {'Default'};
0211         pl = getDefaultPlist;
0212     end
0213     % Build info object
0214     ii = minfo(mfilename, 'ao', '', utils.const.categories.mdc01, '$Id:
mdc1_ifo2control.m,v 1.8 2008/09/05 14:15:14 hewitson Exp $', sets, pl);
0215 end
0216
0217 %-----
0218 % Get Default Plist
0219 %-----
0220 function pl_default = getDefaultPlist()
0221     pl_default = plist();
0222 end
0223 % END

```


mdc1_x2acc

PURPOSE ^

MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter

SYNOPSIS ^

function varargout = mdc1_x2acc(varargin)

DESCRIPTION ^

```
MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter
taking in to account the given spring stiffness.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MDC1_X2ACC converts the input time-series to acceleration with a time-domain
filter
              taking in to account the given spring stiffness.

CALL:        o1dd = mdc1_x2acc(o1, pl)

PARAMETERS: 'Omega2' - the square of the stiffness term [default: -1.3e-6]

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('mdc1_x2acc')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mdc1_x2acc', 'None')

VERSION:     $Id: mdc1_x2acc.m,v 1.7 2008/09/05 11:05:29 ingo Exp $

HISTORY:     11-04-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [sqrt](#) SQRT computes the square root of the data in the AO.

This function is called by:

-  [mdc1_ifo2acc_inloop](#) MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter
0002 % taking in to account the given spring stiffness.
```



```

0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: MDC1_X2ACC converts the input time-series to acceleration with a time-
domain filter
0006 %           taking in to account the given spring stiffness.
0007 %
0008 % CALL:      o1dd = mdc1_x2acc(o1, pl)
0009 %
0010 % PARAMETERS: 'Omega2' - the square of the stiffness term [default: -1.3e-6]
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %           >> ao.getInfo('mdc1_x2acc')
0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> ao.getInfo('mdc1_x2acc', 'None')
0017 %
0018 % VERSION:   $Id: mdc1_x2acc.m,v 1.7 2008/09/05 11:05:29 ingo Exp $
0019 %
0020 % HISTORY:   11-04-08 M Hewitson
0021 %           Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = mdc1_x2acc(varargin)
0026
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 import utils.const.*
0034 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0035
0036 % Collect input variable names
0037 in_names = cell(size(varargin));
0038 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0039
0040 % Collect all AOs and plists
0041 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0043
0044 % Decide on a deep copy or a modify
0045 bs = copy(as, nargout);
0046
0047 % combine plists
0048 pl = combine(pl, getDefaultPlist());
0049
0050 %% Extract parameters from plist
0051
0052 % Get parameters out
0053 w = find(pl, 'Omega2');
0054
0055 % Process each input AO
0056 for j=1:length(bs)
0057     %----- build time-domain filter
0058     % need a zero at the stiffness frequency
0059     % - take the absolute value of omega
0060     z1 = pz(sqrt(abs(w))/2/pi);
0061     % add a stabilising pole up near the Nyquist
0062     p1 = pz(0.7*bs(j).data.fs/2);
0063     % build pzmodel
0064     pzm = pzmodel(sqrt(abs(w)), p1, z1);
0065     % build filter
0066     ft = miir(pzm, plist('fs', bs(j).data.fs));
0067     % filter data forwards and backwards to get f^2 response
0068     bs(j).filtfilt(ft);
0069     % Add history
0070     bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0071     % Set AO name
0072     bs(j).setName(sprintf('%s(%s)', mfilename, ao_invars{j}), 'internal');
0073 end
0074
0075 % Set output
0076 if nargout > 0
0077     varargout{1} = bs;
0078 end
0079 end
0080
0081 %-----
0082 % Get Info Object
0083 %-----
0084 function ii = getInfo(varargin)
0085     if nargin == 1 && strcmpi(varargin{1}, 'None')
0086         sets = {};
0087         pl = [];

```

```
0088     else
0089         sets = {'Default'};
0090         pl    = getDefaultPlist;
0091     end
0092     % Build info object
0093     ii = minfo(mfilename, 'ao', '', utils.const.categories.mdc01, '$Id: mdc1_x2acc.m,v 1.7
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0094 end
0095
0096 %-----
0097 % Get Default Plist
0098 %-----
0099 function pl_default = getDefaultPlist()
0100     pl_default = plist('Omega2', 1.3e-6);
0101 end
0102
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

abs

PURPOSE ^

ABS overloads the Absolute value method for Analysis objects.

SYNOPSIS ^

function varargout = abs(varargin)

DESCRIPTION ^

```
ABS overloads the Absolute value method for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: ABS overloads the Absolute value operator for Analysis objects.

CALL:      ao_out = abs(ao_in);
           ao_out = abs(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('abs')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('abs', 'None')

VERSION:    $Id: abs.m,v 1.33 2008/09/06 15:23:19 hewitson Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [diff](#) DIFF differentiates the data in AO.
-  [dropduplicates](#) DROPDUPLICATES drops all duplicate samples in time-series AOs.
-  [iplot](#) IPLOT provides an intelligent plotting tool for LTPDA.
-  [mdc1_ifo2acc_fd](#) MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
-  [mdc1_x2acc](#) MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter
-  [mltfe](#) MLTFE compute log-frequency space TF
-  [spikecleaning](#) spikecleaning detects and corrects possible spikes in analysis objects
-  [welch](#) WELCH Welch spectral estimation method.
-  [xspect](#) XSPEC performs cross-spectral analysis of various forms.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl_default = getDefaultPlist()

SOURCE CODE ^

```

0001 % ABS overloads the Absolute value method for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ABS overloads the Absolute value operator for Analysis objects.
0005 %
0006 % CALL:          ao_out = abs(ao_in);
0007 %              ao_out = abs(ao_in, pl);
0008 %
0009 % PARAMETERS:    see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO:  Get information about this methods by calling
0012 %              >> ao.getInfo('abs')
0013 %
0014 %              Get information about a specified set-plist by calling:
0015 %              >> ao.getInfo('abs', 'None')
0016 %
0017 % VERSION:      $Id: abs.m,v 1.33 2008/09/06 15:23:19 hewitson Exp $
0018 %
0019 % HISTORY:      12-03-07 M Hewitson
0020 %              Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = abs(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Get default parameters
0041 pl = combine(pl, getDefaultPlist);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargsout);
0045
0046 % Apply method to all AOs
0047 applymethod(bs, ao_invars, 'abs', pl, getDefaultPlist, getInfo);
0048
0049 % Set units
0050 for ii = 1:numel(bs)
0051     app_axis = pl.find('axis');
0052     if ~isempty(find('X'==upper(app_axis), 1))
0053         bs(ii).setXunits(feval('abs', bs(ii).data.xunits), 'internal');
0054     end
0055     if ~isempty(find('Y'==upper(app_axis), 1))
0056         bs(ii).setYunits(feval('abs', bs(ii).data.yunits), 'internal');
0057     end
0058 end
0059
0060 % Set output
0061 if nargsout > 0
0062     varargout{1} = bs;
0063 end
0064 end
0065
0066 %-----
0067 % Get Info Object
0068 %-----
0069 function ii = getInfo(varargin)
0070 if nargin == 1 && strcmpi(varargin{1}, 'None')
0071     sets = {};
0072     pl = [];
0073 else
0074     sets = {'Default'};
0075     pl = getDefaultPlist;
0076 end
0077 % Build info object
0078 ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: abs.m,v 1.33
2008/09/06 15:23:19 hewitson Exp $', sets, pl);
0079 end
0080

```

```
0081 %-----  
0082 % Get Default Plist  
0083 %-----  
0084 function pl_default = getDefaultPlist()  
0085  
0086     pl_default = plist('axis', 'y');  
0087 end  
0088
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

complex

PURPOSE ^

COMPLEX overloads the complex operator for Analysis objects.

SYNOPSIS ^

function varargout = complex(varargin)

DESCRIPTION ^

```
COMPLEX overloads the complex operator for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COMPLEX overloads the complex operator for Analysis objects.
              A3 = COMPLEX(A1,A2) returns the complex result A1 + A2i,
              where A1 and A2 are identically sized real arrays.

CALL:        ao_out = complex(a1, a2);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('complex')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('complex', 'None')

VERSION:      $Id: complex.m,v 1.22 2008/09/05 14:13:27 hewitson Exp $

HISTORY:      20-08-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [complex](#) COMPLEX overloads the complex operator for Analysis objects.
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [complex](#) COMPLEX overloads the complex operator for Analysis objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % COMPLEX overloads the complex operator for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COMPLEX overloads the complex operator for Analysis objects.
0005 %              A3 = COMPLEX(A1,A2) returns the complex result A1 + A2i,
0006 %              where A1 and A2 are identically sized real arrays.
0007 %
0008 % CALL:        ao_out = complex(a1, a2);
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
```

```

0011 %                >> ao.getInfo('complex')
0012 %
0013 %                Get information about a specified set-plist by calling:
0014 %                >> ao.getInfo('complex', 'None')
0015 %
0016 % VERSION:      $Id: complex.m,v 1.22 2008/09/05 14:13:27 hewitson Exp $
0017 %
0018 % HISTORY:      20-08-2007 Diepholz
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%
0022 %
0023 function varargout = complex(varargin)
0024 %
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033 %
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037 %
0038 % Collect all AOs and plists
0039 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040 %
0041 % Decide on a deep copy or a modify
0042 bs = copy(as, nargsout);
0043 %
0044 % Check input arguments number
0045 if length(bs) ~= 2
0046     error('### Incorrect inputs. Please enter 2 AOs');
0047 end
0048 %
0049 % Only support data2D or cdata for now
0050 if isa(bs(1).data, 'data3D') || isa(bs(2).data, 'data3D')
0051     error('### 3D data objects are currently not supported.');
```

```

0052 end
0053 %
0054 % Check for the same data.
0055 if ~strcmp(class(bs(1).data), class(bs(2).data))
0056     error('### The data class of the two AOs must be the same. (%s <-> %s)', ...
0057         class(bs(1).data), class(bs(2).data));
0058 end
0059 %
0060 % Check for the same sample rate
0061 fields = fieldnames(bs(1).data);
0062 if ismember('fs', fields)
0063     if bs(1).data.fs ~= bs(2).data.fs
0064         error('### The sample rate of the two AOs is not the same. Please resample one of
0065 them.');
```

```

0066     end
0067 end
0068 %
0069 % Check the length of the AO's
0070 if length(bs(1).data.getY) ~= length(bs(2).data.getY) || length(bs(1).data.getX) ~=
length(bs(2).data.getX)
0071     error('### The length of the data vectors must be the same.');
```

```

0072 end
0073 %
0074 % The x vector should be the same
0075 if ~isequal(bs(1).data.getX, bs(2).data.getX)
0076     error('### The two data series should have the same x values.');
```

```

0077 end
0078 %
0079 % The time should be the same
0080 if ismember('t0', fields)
0081     if bs(1).data.t0.utc_epoch_milli ~= bs(2).data.t0.utc_epoch_milli
0082         error('### The two data series don't start at the same time.');
```

```

0083     end
0084 end
0085 %
0086 % The x units should match
0087 if ismember('xunits', fields)
0088     if bs(1).data.xunits ~= bs(2).data.xunits
0089         error('### The two data series should have the same x units.');
```

```

0090     end
0091 end
0092 %
0093 % Copy object 1 for the output
0094 bs(1).data.setY(complex(bs(1).data.getY, bs(2).data.getY));
if ismember('yunits', fields)

```

```

0095     if bs(1).data.yunits == bs(2).data.yunits
0096         bs(1).setYunits(bs(1).data.yunits, 'internal');
0097     else
0098         error('### Can't combine data with different units');
0099     end
0100 end
0101
0102 % Add history
0103 bs(1).addHistory(getInfo, getDefaultPlist, ao_invars, [bs(1).hist bs(2).hist]);
0104
0105 % Set name
0106 bs(1).setName(sprintf('complex(%s, %s)', ao_invars{1}, ao_invars{2}), 'internal');
0107
0108 % Set output
0109 varargout{1} = bs(1);
0110 end
0111
0112 %-----
0113 % Get Info Object
0114 %-----
0115 function ii = getInfo(varargin)
0116     if nargin == 1 && strcmpi(varargin{1}, 'None')
0117         sets = {};
0118         pl = [];
0119     else
0120         sets = {'Default'};
0121         pl = getDefaultPlist;
0122     end
0123     % Build info object
0124     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: complex.m,v 1.22
2008/09/05 14:13:27 hewitson Exp $', sets, pl);
0125 end
0126
0127 %-----
0128 % Get Default Plist
0129 %-----
0130 function pl = getDefaultPlist()
0131     pl = plist();
0132 end
0133

```


conj

PURPOSE ^

CONJ overloads the conjugate operator for Analysis objects.

SYNOPSIS ^

function varargout = conj(varargin)

DESCRIPTION ^

```
CONJ overloads the conjugate operator for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CONJ overloads the conjugate operator for Analysis objects.

CALL:      a_conj = conj

VERSION:    $Id: conj.m,v 1.23 2008/09/05 11:15:19 ingo Exp $

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('conj')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('conj', 'None')

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [computeperiodogram](#) COMPUTEPERIODOGRAM Periodogram spectral estimation.
-  [mdc1_ifo2acc_fd](#) MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
-  [mlpsd_m](#) MLPSD_M m-file only version of the LPSD algorithm

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl_default = getDefaultPlist()

SOURCE CODE ^

```
0001 % CONJ overloads the conjugate operator for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CONJ overloads the conjugate operator for Analysis objects.
0005 %
0006 % CALL:      a_conj = conj
0007 %
```

```

0008 % VERSION:      $Id: conj.m,v 1.23 2008/09/05 11:15:19 ingo Exp $
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %      >> ao.getInfo('conj')
0012 %
0013 %      Get information about a specified set-plist by calling:
0014 %      >> ao.getInfo('conj', 'None')
0015 %
0016 % HISTORY:      12-03-07 M Hewitson
0017 %      Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = conj(varargin)
0022
0023 % Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 import utils.const.*
0030 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs and plists
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Combine plists
0044 pl = combine(pl, getDefaultPlist());
0045
0046 % Apply method to all AOs
0047 applymethod(bs, ao_invars, 'conj', pl, getDefaultPlist(), getInfo());
0048
0049 % Reshape the ouput to the same size of the input
0050 if nargout > 0
0051     varargout{1} = bs;
0052 end
0053 end
0054
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl = [];
0062     else
0063         sets = {'Default'};
0064         pl = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: conj.m,v 1.23
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0068 end
0069
0070 %-----
0071 % Get Default Plist
0072 %-----
0073 function pl_default = getDefaultPlist()
0074     pl_default = plist('axis', 'y');
0075 end
0076

```

ctranspose

PURPOSE ^

CTRANSPOSE overloads the ' operator for Analysis Objects.

SYNOPSIS ^

function varargout = ctranspose(varargin)

DESCRIPTION ^

```
CTRANSPOSE overloads the ' operator for Analysis Objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CTRANSPOSE overloads the ' operator for Analysis Objects.
CALL:      a = a1'      % only with data = cdata
VERSION:    $Id: ctranspose.m,v 1.25 2008/09/05 11:15:19 ingo Exp $
PARAMETERS: see help for data2D/applymethod for additional parameters
M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('ctranspose')
              Get information about a specified set-plist by calling:
              >> ao.getInfo('ctranspose', 'None')

HISTORY: 08-05-07 A Monsky
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CTRANSPOSE overloads the ' operator for Analysis Objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CTRANSPOSE overloads the ' operator for Analysis Objects.
0005 %
0006 % CALL:      a = a1'      % only with data = cdata
0007 %
0008 % VERSION:    $Id: ctranspose.m,v 1.25 2008/09/05 11:15:19 ingo Exp $
0009 %
0010 % PARAMETERS: see help for data2D/applymethod for additional parameters
```

```

0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %         >> ao.getInfo('ctranpose')
0014 %
0015 %         Get information about a specified set-plist by calling:
0016 %         >> ao.getInfo('ctranpose', 'None')
0017 %
0018 % HISTORY: 08-05-07 A Monsky
0019 %         Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = ctranpose(varargin)
0024 %
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033 %
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037 %
0038 % Collect all AOs
0039 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0041 %
0042 % Decide on a deep copy or a modify
0043 bs = copy(as, nargout);
0044 %
0045 % Combine plists
0046 pl = combine(pl, getDefaultPlist);
0047 %
0048 axis = find(pl, 'axis');
0049 %
0050 for kk = 1:numel(bs)
0051     % Special case for cdata-objects because this objects doesn't have x values
0052     if isa(bs(kk).data, 'data2D')
0053         x = bs(kk).data.x;
0054         y = bs(kk).data.y;
0055         % Set the y-value to an empty array because the y-data keeps always the same
0056         % shape
0057         bs(kk).data.setY([]);
0058         bs(kk).data.setX([]);
0059         switch upper(axis)
0060             case 'X'
0061                 bs(kk).data.setX(x');
0062                 bs(kk).data.setY(y');
0063             case 'Y'
0064                 bs(kk).data.setX(x');
0065                 bs(kk).data.setY(y');
0066             case 'XY'
0067                 bs(kk).data.setX(x');
0068                 bs(kk).data.setY(y');
0069             otherwise
0070                 error('### Unknown axis to operate on.');

```

```
0097 %-----
0098 function ii = getInfo(varargin)
0099
0100     if nargin == 1 && strcmpi(varargin{1}, 'None')
0101         sets = {};
0102         pl    = [];
0103     else
0104         sets = {'Default'};
0105         pl    = getDefaultPlist;
0106     end
0107     % Build info object
0108     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: ctranspose.m,v 1.25
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0109 end
0110
0111 %-----
0112 % Get Default Plist
0113 %-----
0114 function pl_default = getDefaultPlist()
0115     pl_default = plist('axis', 'xy');
0116 end
0117
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

det

PURPOSE ^

DET overloads the determinant function for Analysis objects.

SYNOPSIS ^

function varargout = det(varargin)

DESCRIPTION ^

```
DET overloads the determinant function for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DET overloads the determinant function for Analysis objects.

CALL:      a = det(a1)    % only with cdata AOs

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('det')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('det', 'None')

VERSION:    $Id: det.m,v 1.24 2008/09/05 11:15:19 ingo Exp $

HISTORY:    08-05-07 A Monsky
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [mchol](#) MCHOL

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```
0001 % DET overloads the determinant function for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DET overloads the determinant function for Analysis objects.
0005 %
0006 % CALL:      a = det(a1)    % only with cdata AOs
0007 %
0008 % PARAMETERS: see help for data2D/applymethod for additional parameters
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
```

```

0011 %                >> ao.getInfo('det')
0012 %
0013 %                Get information about a specified set-plist by calling:
0014 %                >> ao.getInfo('det', 'None')
0015 %
0016 % VERSION:      $Id: det.m,v 1.24 2008/09/05 11:15:19 ingo Exp $
0017 %
0018 % HISTORY: 08-05-07 A Monsky
0019 %                Creation
0020 %
0021 %%%%%%%%%%%
0022 %
0023 function varargout = det(varargin)
0024
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038 % Collect all AOs
0039 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040
0041 % Decide on a deep copy or a modify
0042 bs = copy(as, nargout);
0043
0044 % go through analysis objects
0045 for j=1: numel(bs)
0046     if isa(bs(j).data, 'cdata')
0047         if size(bs(j).data.y,1) ~= size(bs(j).data.y,2)
0048             error('### The y data must be a square matrix.')
0049         end
0050         % Apply method to all AOs
0051         applymethod(bs(j), ao_invars(j), 'det', plist, getDefaultPlist, getInfo);
0052     else
0053         error('### this function works for cdata type AO only')
0054     end
0055 end
0056
0057 % Set outputs
0058 if nargout > 0
0059     varargout{1} = bs;
0060 end
0061 end
0062
0063 %-----
0064 % Get Info Object
0065 %-----
0066 function ii = getInfo(varargin)
0067
0068 if nargin == 1 && strcmpi(varargin{1}, 'None')
0069     sets = {};
0070     pl = [];
0071 else
0072     sets = {'Default'};
0073     pl = getDefaultPlist;
0074 end
0075 % Build info object
0076 ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: det.m,v 1.24
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0077 end
0078
0079 %-----
0080 % Get Default Plist
0081 %-----
0082 function pl = getDefaultPlist()
0083     pl = plist();
0084 end
0085

```

diag

PURPOSE ^

DIAG overloads the diagonal operator for Analysis Objects.

SYNOPSIS ^

function varargout = diag(varargin)

DESCRIPTION ^

```
DIAG overloads the diagonal operator for Analysis Objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DIAG overloads the diagonal operator for Analysis Objects.

CALL:      b = diag (a)      % only with data = cdata
           b = diag (a, pl)

INPUTS:    a - input analysis object
           pl - parameter list

OUTPUTS:   b - output analysis object

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('diag')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('diag', 'None')

VERSION:   $Id: diag.m,v 1.25 2008/09/05 11:15:19 ingo Exp $

HISTORY:   08-05-07 A Monsky
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

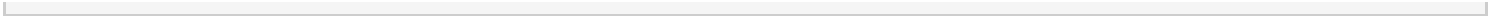
```
0001 % DIAG overloads the diagonal operator for Analysis Objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DIAG overloads the diagonal operator for Analysis Objects.
0005 %
0006 % CALL:      b = diag (a)      % only with data = cdata
0007 %           b = diag (a, pl)
```



```

0008 %
0009 % INPUTS:      a - input analysis object
0010 %              pl - parameter list
0011 %
0012 % OUTPUTS:     b - output analysis object
0013 %
0014 % PARAMETERS:  see help for data2D/applymethod for additional parameters
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 %              >> ao.getInfo('diag')
0018 %
0019 %              Get information about a specified set-plist by calling:
0020 %              >> ao.getInfo('diag', 'None')
0021 %
0022 % VERSION:     $Id: diag.m,v 1.25 2008/09/05 11:15:19 ingo Exp $
0023 %
0024 % HISTORY:     08-05-07 A Monsky
0025 %              Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = diag(varargin)
0030
0031 % Check if this is a call for parameters
0032 if utils.helper.isinfocall(varargin{:})
0033     varargout{1} = getInfo(varargin{3});
0034     return
0035 end
0036
0037 import utils.const.*
0038 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0039
0040 % Collect input variable names
0041 in_names = cell(size(varargin));
0042 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0043
0044 % Collect all AOs
0045 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0046 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0047
0048 % Decide on a deep copy or a modify
0049 bs = copy(as, nargout);
0050
0051 % Combine plists
0052 pl = combine(pl, getDefaultPlist);
0053
0054 %% go through analysis objects
0055 for j=1:numel(bs)
0056     if ~isa(bs(j).data, 'cdata')
0057         warning('!!! This methods works for cdata type AO only. Skipping AO %s',
0058             ao_invars{j})
0059     else
0060         % Apply method to all AOs
0061         applymethod(bs(j), ao_invars(j), 'diag', pl, getDefaultPlist, getInfo);
0062     end
0063 end
0064
0065 % Set output
0066 if nargout > 0
0067     varargout{1} = bs;
0068 end
0069
0070 %-----
0071 % Get Info Object
0072 %-----
0073 function ii = getInfo(varargin)
0074     if nargin == 1 && strcmpi(varargin{1}, 'None')
0075         sets = {};
0076         pl = [];
0077     else
0078         sets = {'Default'};
0079         pl = getDefaultPlist;
0080     end
0081     % Build info object
0082     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: diag.m,v 1.25
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0083 end
0084
0085 %-----
0086 % Get Default Plist
0087 %-----
0088 function pl = getDefaultPlist()
0089     pl = plist('option', 0);
0090 end
0091

```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

eig

PURPOSE ^

EIG overloads the determinant function for Analysis objects.

SYNOPSIS ^

function varargout = eig(varargin)

DESCRIPTION ^

```
EIG overloads the determinant function for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: EIG overloads the determinant function for Analysis objects.

CALL:      e = eig (a,pl)    % only with data = cdata

INPUTS:    pl   - a parameter list
            a    - input analysis object

OUTPUTS:    like matlab function

PARAMETERS: 'option' - a string or value that can be submitted
              i.e. 'nobalance' to disable balancing options are the
                  same as for the matlab function.

REMARKS:    See help for data2D/applymethod for additional parameters.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('eig')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('eig', 'None')

VERSION:    $Id: eig.m,v 1.26 2008/09/05 11:15:19 ingo Exp $

HISTORY:    08-05-07 A Monsky
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % EIG overloads the determinant function for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: EIG overloads the determinant function for Analysis objects.
```

```

0005 %
0006 % CALL:      e = eig (a,pl)    % only with data = cdata
0007 %
0008 % INPUTS:    pl    - a parameter list
0009 %            a     - input analysis object
0010 %
0011 % OUTPUTS:    like matlab function
0012 %
0013 % PARAMETERS: 'option' - a string or value that can be submitted
0014 %              i.e. 'nobalance' to disable balancing options are the
0015 %              same as for the matlab function.
0016 %
0017 % REMARKS:    See help for data2D/applymethod for additional parameters.
0018 %
0019 % M-FILE INFO: Get information about this methods by calling
0020 %              >> ao.getInfo('eig')
0021 %
0022 %              Get information about a specified set-plist by calling:
0023 %              >> ao.getInfo('eig', 'None')
0024 %
0025 % VERSION:    $Id: eig.m,v 1.26 2008/09/05 11:15:19 ingo Exp $
0026 %
0027 % HISTORY:    08-05-07 A Monsky
0028 %              Creation
0029 %
0030 %%%%%%%%%%%
0031 %
0032 % These options removed in the recoding. If we really need this feature
0033 % then I say that this method can't use the ao/applymethod() method, but
0034 % must be coded up individually.
0035 %          [v, d] = eig (a,pl)
0036 %          [v, d] = eig (a)
0037 %
0038 function varargout = eig(varargin)
0039 %
0040 % Check if this is a call for parameters
0041 if utils.helper.isinfocall(varargin{:})
0042     varargout{1} = getInfo(varargin{3});
0043     return
0044 end
0045 %
0046 import utils.const.*
0047 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0048 %
0049 % Collect input variable names
0050 in_names = cell(size(varargin));
0051 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0052 %
0053 % Collect all AOs
0054 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0055 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0056 %
0057 % Decide on a deep copy or a modify
0058 bs = copy(as, nargout);
0059 %
0060 % Combine plists
0061 pl = combine(pl, getDefaultPlist);
0062 %
0063 %% go through analysis objects
0064 for j=1:numel(bs)
0065     if isa(bs(j).data, 'cdata')
0066         if size(bs(j).data.y,1) ~= size(bs(j).data.y,2)
0067             error('### The y data must be a square matrix.')
0068         end
0069         % Apply method to all AOs
0070         applymethod(bs(j), ao_invars(j), 'eig', pl, getDefaultPlist, getInfo);
0071     else
0072         error('### this function works for cdata type AO only')
0073     end
0074 end
0075 %
0076 % Set output
0077 if nargout > 0
0078     varargout{1} = bs;
0079 end
0080 end
0081 %-----
0082 % Get Info Object
0083 %-----
0084 %
0085 function ii = getInfo(varargin)
0086 if nargin == 1 && strcmpi(varargin{1}, 'None')
0087     sets = {};
0088     pl    = [];
0089 else
0090     sets = {'Default'};

```

```
0091     pl    = getDefaultPlist;
0092     end
0093     % Build info object
0094     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: eig.m,v 1.26
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0095     end
0096
0097     %-----
0098     % Get Default Plist
0099     %-----
0100     function pl = getDefaultPlist()
0101         pl = plist('option', '');
0102     end
0103
0104
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

exp

PURPOSE ^

EXP overloads the exp operator for Analysis objects. Exponential.

SYNOPSIS ^

function varargout = exp (varargin)

DESCRIPTION ^

```
EXP overloads the exp operator for Analysis objects. Exponential.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: EXP overloads the exp operator for Analysis objects. Exponential.
              EXP(ao) is the exponential of the elements of ao.data
              e to the ao.data.

CALL:        ao_out = exp(ao_in);
              ao_out = exp(ao_in, pl);
              ao_out = exp(ao1, pl1, ao_vector, ao_matrix, pl2);

POSSIBLE VALUES: ao_in  = [ao2 ao3]
                  ao_in  = ao_vector
                  ao_in  = ao_matrix

PARAMETERS:   see help for data2D/applymethod for additional parameters

M-FILE INFO:  Get information about this methods by calling
              >> ao.getInfo('exp')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('exp', 'None')

VERSION:      $Id: exp.m,v 1.23 2008/09/05 14:11:47 hewitson Exp $

HISTORY:      23-05-2007 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [computeDFT](#) COMPUTEDFT Computes DFT using FFT or Goertzel
-  [dft](#) DFT computes the DFT of the input time-series at the requested frequencies.
-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input PSD.
-  [mlpsd_m](#) MLPSD_M m-file only version of the LPSD algorithm
-  [mlpsd_mex](#) MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD
-  [mltfe](#) MLTFE compute log-frequency space TF
-  [search](#) SEARCH selects AOs that match the given name.

SUBFUNCTIONS ^

 function ii = getInfo(varargin)
 function pl_default = getDefaultPlist()

SOURCE CODE ^

```

0001 % EXP overloads the exp operator for Analysis objects. Exponential.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: EXP overloads the exp operator for Analysis objects. Exponential.
0005 %               EXP(ao) is the exponential of the elements of ao.data
0006 %               e to the ao.data.
0007 %
0008 % CALL:         ao_out = exp(ao_in);
0009 %               ao_out = exp(ao_in, pl);
0010 %               ao_out = exp(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % POSSIBLE VALUES: ao_in = [ao2 ao3]
0013 %                   ao_in = ao_vector
0014 %                   ao_in = ao_matrix
0015 %
0016 % PARAMETERS:   see help for data2D/applymethod for additional parameters
0017 %
0018 % M-FILE INFO:  Get information about this methods by calling
0019 %               >> ao.getInfo('exp')
0020 %
0021 %               Get information about a specified set-plist by calling:
0022 %               >> ao.getInfo('exp', 'None')
0023 %
0024 % VERSION:      $Id: exp.m,v 1.23 2008/09/05 14:11:47 hewitson Exp $
0025 %
0026 % HISTORY:      23-05-2007 Diepholz
0027 %               Creation
0028 %
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 function varargout = exp (varargin)
0032
0033 % Check if this is a call for parameters
0034 if utils.helper.isinfocall(varargin{:})
0035     varargout{1} = getInfo(varargin{3});
0036     return
0037 end
0038
0039 % Collect input variable names
0040 in_names = cell(size(varargin));
0041 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0042
0043 % Collect all AOs
0044 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0045 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0046
0047 % Decide on a deep copy or a modify
0048 bs = copy(as, nargout);
0049
0050 % Combine plists
0051 pl = combine(pl, getDefaultPlist);
0052
0053 % Apply method to all AOs
0054 applymethod(bs, ao_invars, 'exp', pl, getDefaultPlist, getInfo);
0055
0056 % Set units
0057 for ii = 1:numel(bs)
0058     app_axis = pl.find('axis');
0059     if ~isempty(find('X'==upper(app_axis), 1))
0060         bs(ii).setXunits(feval('exp', bs(ii).data.xunits), 'internal');
0061     end
0062     if ~isempty(find('Y'==upper(app_axis), 1))
0063         bs(ii).setYunits(feval('exp', bs(ii).data.yunits), 'internal');
0064     end
0065 end
0066
0067 % Reshape the ouput to the same size of the input
0068 if nargout > 0
0069     varargout{1} = bs;
0070 end
0071 end
0072
0073 %-----
0074 % Get Info Object
0075 %-----
0076 function ii = getInfo(varargin)
0077 if nargin == 1 && strcmpi(varargin{1}, 'None')
```

```
0078     sets = {};  
0079     pl    = [];  
0080     else  
0081         sets = {'Default'};  
0082         pl    = getDefaultPlist;  
0083     end  
0084     % Build info object  
0085     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: exp.m,v 1.23  
2008/09/05 14:11:47 hewitson Exp $', sets, pl);  
0086 end  
0087  
0088 %-----  
0089 % Get Default Plist  
0090 %-----  
0091 function pl_default = getDefaultPlist()  
0092     pl_default = plist('axis', 'y');  
0093 end  
0094
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

imag

PURPOSE ^

IMAG overloads the imaginary operator for Analysis objects.

SYNOPSIS ^

function varargout = imag(varargin)

DESCRIPTION ^

```
IMAG overloads the imaginary operator for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: IMAG overloads the imaginary operator for Analysis objects.

CALL:      ao = imag(ao_in);
           ao = imag(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('imag')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('imag', 'None')

VERSION:    $Id: imag.m,v 1.12 2008/09/05 11:15:19 ingo Exp $

HISTORY:    20-0802007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [export](#) EXPORT export an analysis object to a text file.
-  [iplot](#) IPLOT provides an intelligent plotting tool for LTPDA.
-  [plot](#) PLOT a simple plot of analysis objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % IMAG overloads the imaginary operator for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: IMAG overloads the imaginary operator for Analysis objects.
0005 %
0006 % CALL:      ao = imag(ao in);
```

```

0007 %          ao = imag(ao_in, pl);
0008 %
0009 % PARAMETERS:  see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO:  Get information about this methods by calling
0012 %               >> ao.getInfo('imag')
0013 %
0014 %               Get information about a specified set-plist by calling:
0015 %               >> ao.getInfo('imag', 'None')
0016 %
0017 % VERSION:      $Id: imag.m,v 1.12 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:      20-0802007 Diepholz
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = imag(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 import utils.const.*
0033 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs and plists
0040 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0041 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargsout);
0045
0046 % Combine plists
0047 pl = combine(pl, getDefaultPlist);
0048
0049 % Apply method to all AOs
0050 applymethod(bs, ao_invars, 'imag', pl, getDefaultPlist, getInfo);
0051
0052 % Reshape the ouput to the same size of the input
0053 if nargsout > 0
0054     varargout{1} = bs;
0055 end
0056
0057 end
0058 %-----
0059 % Get Info Object
0060 %-----
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl    = [];
0065     else
0066         sets = {'Default'};
0067         pl    = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: imag.m,v 1.12
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0071 end
0072
0073 %-----
0074 % Get Default Plist
0075 %-----
0076 function pl_default = getDefaultPlist()
0077     pl_default = plist('axis', 'y');
0078 end
0079

```

inv

PURPOSE ^

INV overloads the inverse function for Analysis Objects.

SYNOPSIS ^

function varargout = inv(varargin)

DESCRIPTION ^

```
INV overloads the inverse function for Analysis Objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: INV overloads the inverse function for Analysis Objects.

CALL:      a = inv(a1)    % only with data = cdata

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('inv')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('inv', 'None')

VERSION:    $Id: inv.m,v 1.22 2008/09/05 11:15:19 ingo Exp $

HISTORY:    08-05-07 A Monsky
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % INV overloads the inverse function for Analysis Objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: INV overloads the inverse function for Analysis Objects.
0005 %
0006 % CALL:      a = inv(a1)    % only with data = cdata
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> ao.getInfo('inv')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> ao.getInfo('inv', 'None')
0013 %
0014 % VERSION:    $Id: inv.m,v 1.22 2008/09/05 11:15:19 ingo Exp $
0015 %
```

```

0016 % HISTORY:      08-05-07 A Monsky
0017 %
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = inv(varargin)
0022
0023 % Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 import utils.const.*
0030 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038
0039 % Decide on a deep copy or a modify
0040 bs = copy(as, nargout);
0041
0042 %% go through analysis objects
0043 for j=1: numel(bs)
0044     if isa(bs(j).data, 'cdata')
0045         if size(bs(j).data.y,1) ~= size(bs(j).data.y,2)
0046             error('### The y data must be a square matrix.')
0047         end
0048         % Apply method to all AOs
0049         applymethod(bs(j), ao_invars(j), 'inv', plist, getDefaultPlist, getInfo);
0050     else
0051         error('### this function works for cdata type AO only')
0052     end
0053 end
0054
0055 % Set outputs
0056 if nargout > 0
0057     varargout{1} = bs;
0058 end
0059 end
0060
0061 %-----
0062 % Get Info Object
0063 %-----
0064 function ii = getInfo(varargin)
0065     if nargin == 1 && strcmpi(varargin{1}, 'None')
0066         sets = {};
0067         pl = [];
0068     else
0069         sets = {'Default'};
0070         pl = getDefaultPlist;
0071     end
0072     % Build info object
0073     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: inv.m,v 1.22
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0074 end
0075
0076 %-----
0077 % Get Default Plist
0078 %-----
0079 function pl = getDefaultPlist()
0080     pl = plist();
0081 end
0082
0083

```

ln

PURPOSE ^

LN overloads the log operator for Analysis objects. Natural logarithm.

SYNOPSIS ^

function varargout = ln (varargin)

DESCRIPTION ^

```
LN overloads the log operator for Analysis objects. Natural logarithm.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LN overloads the log operator for Analysis objects.
              Natural logarithm.
              LN(ao) is the natural logarithm of the elements of ao.data.

CALL:        ao_out = ln(ao_in);
              ao_out = ln(ao_in, pl);
              ao_out = ln(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS:  see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('ln')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('ln', 'None')

VERSION:     $Id: ln.m,v 1.17 2008/09/05 14:11:47 hewitson Exp $

HISTORY:     23-05-2007 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

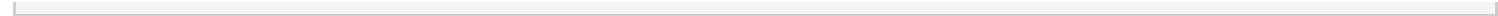
SOURCE CODE ^

```
0001 % LN overloads the log operator for Analysis objects. Natural logarithm.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LN overloads the log operator for Analysis objects.
0005 %              Natural logarithm.
0006 %              LN(ao) is the natural logarithm of the elements of ao.data.
0007 %
0008 % CALL:        ao_out = ln(ao_in);
0009 %              ao_out = ln(ao_in, pl);
```

```

0010 %          ao_out = ln(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % PARAMETERS:  see help for data2D/applymethod for additional parameters
0013 %
0014 % M-FILE INFO:  Get information about this methods by calling
0015 %               >> ao.getInfo('ln')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('ln', 'None')
0019 %
0020 % VERSION:      $Id: ln.m,v 1.17 2008/09/05 14:11:47 hewitson Exp $
0021 %
0022 % HISTORY:      23-05-2007 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026 %
0027 function varargout = ln (varargin)
0028
0029     import utils.const.*
0030
0031     % Check if this is a call for parameters
0032     if utils.helper.isinfocall(varargin{:})
0033         varargout{1} = getInfo(varargin{3});
0034         return
0035     end
0036
0037     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0038
0039     % Collect input variable names
0040     in_names = cell(size(varargin));
0041     for ii = 1:nargin, in_names{ii} = inputname(ii); end
0042
0043     % Collect all AOs
0044     [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0045     pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0046
0047     % Decide on a deep copy or a modify
0048     bs = copy(as, nargout);
0049
0050     % Combine plists
0051     pl = combine(pl, getDefaultPlist);
0052
0053     % Apply method to all AOs
0054     applymethod(bs, ao_invars, 'log', pl, getDefaultPlist, getInfo);
0055
0056     % Set units
0057     for ii = 1:numel(bs)
0058         app_axis = pl.find('axis');
0059         if ~isempty(find('X'==upper(app_axis), 1))
0060             bs(ii).setXunits(feval('log', bs(ii).data.xunits), 'internal');
0061         end
0062         if ~isempty(find('Y'==upper(app_axis), 1))
0063             bs(ii).setYunits(feval('log', bs(ii).data.yunits), 'internal');
0064         end
0065     end
0066
0067     % Reshape the ouput to the same size of the input
0068     if nargout > 0
0069         varargout{1} = bs;
0070     end
0071 end
0072
0073 %-----
0074 % Get Info Object
0075 %-----
0076 function ii = getInfo(varargin)
0077     if nargin == 1 && strcmpi(varargin{1}, 'None')
0078         sets = {};
0079         pl = [];
0080     else
0081         sets = {'Default'};
0082         pl = getDefaultPlist;
0083     end
0084     % Build info object
0085     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: ln.m,v 1.17
2008/09/05 14:11:47 hewitson Exp $', sets, pl);
0086 end
0087
0088 %-----
0089 % Get Default Plist
0090 %-----
0091 function pl_default = getDefaultPlist()
0092     pl_default = plist('axis', 'y');
0093 end
0094

```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

log

PURPOSE ^

LOG overloads the log operator for Analysis objects. Natural logarithm.

SYNOPSIS ^

function varargout = log (varargin)

DESCRIPTION ^

```
LOG overloads the log operator for Analysis objects. Natural logarithm.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LOG overloads the log operator for Analysis objects.
              Natural logarithm.
              LOG(ao) is the natural logarithm of the elements of ao.data.

CALL:        ao_out = log(ao_in);
              ao_out = log(ao_in, pl);
              ao_out = log(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS:  see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('log')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('log', 'None')

VERSION:     $Id: log.m,v 1.22 2008/09/05 14:11:47 hewitson Exp $

HISTORY:     23-05-2007 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % LOG overloads the log operator for Analysis objects. Natural logarithm.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LOG overloads the log operator for Analysis objects.
0005 %              Natural logarithm.
0006 %              LOG(ao) is the natural logarithm of the elements of ao.data.
0007 %
0008 % CALL:        ao_out = log(ao_in);
0009 %              ao_out = log(ao_in, pl);
```



```

0010 %          ao_out = log(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % PARAMETERS:  see help for data2D/applymethod for additional parameters
0013 %
0014 % M-FILE INFO:  Get information about this methods by calling
0015 %               >> ao.getInfo('log')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('log', 'None')
0019 %
0020 % VERSION:      $Id: log.m,v 1.22 2008/09/05 14:11:47 hewitson Exp $
0021 %
0022 % HISTORY:      23-05-2007 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026 %
0027 function varargout = log (varargin)
0028
0029     import utils.const.*
0030
0031     % Check if this is a call for parameters
0032     if utils.helper.isinfocall(varargin{:})
0033         varargout{1} = getInfo(varargin{3});
0034         return
0035     end
0036
0037     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0038
0039     % Collect input variable names
0040     in_names = cell(size(varargin));
0041     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0042
0043     % Collect all AOs
0044     [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0045     pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0046
0047     % Decide on a deep copy or a modify
0048     bs = copy(as, nargout);
0049
0050     % Combine plists
0051     pl = combine(pl, getDefaultPlist);
0052
0053     % Apply method to all AOs
0054     applymethod(bs, ao_invars, 'log', pl, getDefaultPlist, getInfo);
0055
0056     % Set units
0057     for ii = 1:numel(bs)
0058         app_axis = pl.find('axis');
0059         if ~isempty(find('X'==upper(app_axis), 1))
0060             bs(ii).setXunits(feval('log', bs(ii).data.xunits), 'internal');
0061         end
0062         if ~isempty(find('Y'==upper(app_axis), 1))
0063             bs(ii).setYunits(feval('log', bs(ii).data.yunits), 'internal');
0064         end
0065     end
0066
0067     % Reshape the ouput to the same size of the input
0068     if nargout > 0
0069         varargout{1} = bs;
0070     end
0071 end
0072
0073 %-----
0074 % Get Info Object
0075 %-----
0076 function ii = getInfo(varargin)
0077     if nargin == 1 && strcmpi(varargin{1}, 'None')
0078         sets = {};
0079         pl = [];
0080     else
0081         sets = {'Default'};
0082         pl = getDefaultPlist;
0083     end
0084     % Build info object
0085     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: log.m,v 1.22
2008/09/05 14:11:47 hewitson Exp $', sets, pl);
0086 end
0087
0088 %-----
0089 % Get Default Plist
0090 %-----
0091 function pl_default = getDefaultPlist()
0092     pl_default = plist('axis', 'y');
0093 end
0094

```


Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

log10

PURPOSE ^

LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.

SYNOPSIS ^

function varargout = log10 (varargin)

DESCRIPTION ^

```
LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LOG10 overloads the log10 operator for Analysis objects.
              Common (base 10) logarithm.
              LOG10(ao) is the base 10 logarithm of the elements of ao.data.

CALL:
      ao_out = log10(ao_in);
      ao_out = log10(ao_in, pl);
      ao_out = log10(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS:  see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('log10')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('log10', 'None')

VERSION:     $Id: log10.m,v 1.19 2008/09/05 14:11:47 hewitson Exp $

HISTORY:     23-05-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [fromFSfcn](#) FROMFSFCN Construct an ao from a fs-function string
-  [mdc1_input_noises](#) LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: LOG10 overloads the log10 operator for Analysis objects.
0005 %               Common (base 10) logarithm.
0006 %               LOG10(ao) is the base 10 logarithm of the elements of ao.data.
0007 %
0008 % CALL:         ao_out = log10(ao_in);
0009 %               ao_out = log10(ao_in, pl);
0010 %               ao_out = log10(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % PARAMETERS:   see help for data2D/applymethod for additional parameters
0013 %
0014 % M-FILE INFO:  Get information about this methods by calling
0015 %               >> ao.getInfo('log10')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('log10', 'None')
0019 %
0020 % VERSION:      $Id: log10.m,v 1.19 2008/09/05 14:11:47 hewitson Exp $
0021 %
0022 % HISTORY:      23-05-2007 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%
0026
0027 function varargout = log10 (varargin)
0028
0029     import utils.const.*
0030
0031     % Check if this is a call for parameters
0032     if utils.helper.isinfocall(varargin{:})
0033         varargout{1} = getInfo(varargin{3});
0034         return
0035     end
0036
0037     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0038
0039     % Collect input variable names
0040     in_names = cell(size(varargin));
0041     for ii = 1:nargin, in_names{ii} = inputname(ii); end
0042
0043     % Collect all AOs
0044     [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0045     pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0046
0047     % Decide on a deep copy or a modify
0048     bs = copy(as, nargout);
0049
0050     % Combine plists
0051     pl = combine(pl, getDefaultPlist);
0052
0053     % Apply method to all AOs
0054     applymethod(bs, ao_invars, 'log10', pl, getDefaultPlist, getInfo);
0055
0056     % Set units
0057     for ii = 1:numel(bs)
0058         app_axis = pl.find('axis');
0059         if ~isempty(find('X'==upper(app_axis), 1))
0060             bs(ii).setXunits(feval('log10', bs(ii).data.xunits), 'internal');
0061         end
0062         if ~isempty(find('Y'==upper(app_axis), 1))
0063             bs(ii).setYunits(feval('log10', bs(ii).data.yunits), 'internal');
0064         end
0065     end
0066
0067     % Reshape the ouput to the same size of the input
0068     if nargout > 0
0069         varargout{1} = bs;
0070     end
0071 end
0072
0073 %-----
0074 % Get Info Object
0075 %-----
0076 function ii = getInfo(varargin)
0077     if nargin == 1 && strcmpi(varargin{1}, 'None')
0078         sets = {};
0079         pl = [];
0080     else
0081         sets = {'Default'};
0082         pl = getDefaultPlist;
0083     end
0084     % Build info object
0085     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: log10.m,v 1.19
2008/09/05 14:11:47 hewitson Exp $', sets, pl);
0086 end
0087
0088 %-----

```

```
0089 % Get Default Plist
0090 %-----
0091 function pl_default = getDefaultPlist()
0092     pl_default = plist('axis', 'y');
0093 end
0094
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

max

PURPOSE ^

MAX computes the maximum value of the data in the AO.

SYNOPSIS ^

function varargout = max(varargin)

DESCRIPTION ^

```
MAX computes the maximum value of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MAX computes the maximum value of the data in the AO.

CALL:      ao_out = max(ao_in);
           ao_out = max(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('max')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('max', 'None')

VERSION:    $Id: max.m,v 1.13 2008/09/08 08:31:56 hewitson Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [max](#) MAX computes the maximum value of the data in the AO.

This function is called by:

-  [diff](#) DIFF differentiates the data in AO.
-  [fromDatafile](#) FROMDATAFILE Construct an ao from filename AND parameter list
-  [join](#) JOIN multiple AOs into a single AO.
-  [max](#) MAX computes the maximum value of the data in the AO.
-  [welchparse](#) WELCHPARSE Parser for the PWELCH & SPECTROGRAM functions

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MAX computes the maximum value of the data in the AO.
```

```

0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MAX computes the maximum value of the data in the A0.
0005 %
0006 % CALL:      ao_out = max(ao_in);
0007 %            ao_out = max(ao_in, pl);
0008 %
0009 % PARAMETERS: see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 %               >> ao.getInfo('max')
0013 %
0014 %               Get information about a specified set-plist by calling:
0015 %               >> ao.getInfo('max', 'None')
0016 %
0017 % VERSION:    $Id: max.m,v 1.13 2008/09/08 08:31:56 hewitson Exp $
0018 %
0019 % HISTORY:    12-03-07 M Hewitson
0020 %              Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = max(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Get default parameters
0041 pl = combine(pl, getDefaultPlist);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargout);
0045
0046 % Apply method to all AOs
0047 for j=1:numel(bs)
0048     % get max value
0049     if isfield(bs(j).data, 'x')
0050         switch find(pl, 'axis')
0051             case 'x'
0052                 [mx, idx] = max(bs(j).data.getX);
0053                 my = bs(j).data.getY(idx);
0054             case 'y'
0055                 [my, idx] = max(bs(j).data.getY);
0056                 mx = bs(j).data.getX(idx);
0057         end
0058         bs(j).setXY(mx, my, 'internal');
0059     else
0060         [my, idx] = max(bs(j).data.getY);
0061         bs(j).setY(my, 'internal');
0062     end
0063     % remove fs and nsecs
0064     if ismethod(bs(j).data, 'setFs')
0065         bs(j).setFs(NaN, 'internal');
0066     end
0067     if ismethod(bs(j).data, 'setNsecs')
0068         bs(j).data.setNsecs(NaN, 'internal');
0069     end
0070     % append history
0071     bs(j).addHistory(getInfo('None'), pl, ao_invars(j), bs(j).hist);
0072     % Set new A0 name
0073     bs(j).setName(['max(' ao_invars{j} ')'], 'internal');
0074 end
0075
0076 % Set output
0077 if nargout > 0
0078     varargout{1} = bs;
0079 end
0080
0081
0082 %-----
0083 % Get Info Object
0084 %-----
0085 function ii = getInfo(varargin)
0086     if nargin == 1 && strcmpi(varargin{1}, 'None')
0087         sets = {};

```

```

0088     pl    = [];
0089     else
0090         sets = {'Default'};
0091         pl    = getDefaultPlist;
0092     end
0093     % Build info object
0094     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: max.m,v 1.13
2008/09/08 08:31:56 hewitson Exp $', sets, pl);
0095 end
0096
0097 %-----
0098 % Get Default Plist
0099 %-----
0100 function pl_default = getDefaultPlist()
0101
0102     pl_default = plist('axis', 'y');
0103 end
0104
0105 % END

```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

mean

PURPOSE ^

MEAN computes the mean value of the data in the AO.

SYNOPSIS ^

function varargout = mean(varargin)

DESCRIPTION ^

```
MEAN computes the mean value of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MEAN computes the mean value of the data in the AO.

CALL:      ao_out = mean(ao_in);
           ao_out = mean(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('mean')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('mean', 'None')

VERSION:    $Id: mean.m,v 1.24 2008/09/05 11:15:19 ingo Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [diff](#) DIFF differentiates the data in AO.
-  [dsmean](#) DSMEAN performs a simple downsampling by taking the mean of every N samples.
-  [gapfilling](#) GAPFILLING fills possible gaps in data.
-  [mlpsd_m](#) MLPSD_M m-file only version of the LPSD algorithm
-  [spikecleaning](#) spikecleaning detects and corrects possible spikes in analysis objects

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MEAN computes the mean value of the data in the AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0003 %
0004 % DESCRIPTION: MEAN computes the mean value of the data in the AO.
0005 %
0006 % CALL:          ao_out = mean(ao_in);
0007 %              ao_out = mean(ao_in, pl);
0008 %
0009 % PARAMETERS:    see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO:   Get information about this methods by calling
0012 %               >> ao.getInfo('mean')
0013 %
0014 %               Get information about a specified set-plist by calling:
0015 %               >> ao.getInfo('mean', 'None')
0016 %
0017 % VERSION:       $Id: mean.m,v 1.24 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:       12-03-07 M Hewitson
0020 %                 Creation
0021 %
0022 %%%%%%%%%%%
0023
0024 function varargout = mean(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Apply method to all AOs
0044 applymethod(bs, ao_invars, 'mean', pl, getDefaultPlist, getInfo);
0045
0046 % remove fs and nsecs
0047 for j=1:numel(bs)
0048     if ismethod(bs(j).data, 'setFs')
0049         if isprop(bs(j).data, 'nsecs')
0050             bs(j).setFs(1/bs(j).data.nsecs, 'internal');
0051         else
0052             bs(j).setFs(NaN);
0053         end
0054     end
0055 end
0056
0057 % Set output
0058 if nargout > 0
0059     varargout{1} = bs;
0060 end
0061 end
0062
0063 %-----
0064 % Get Info Object
0065 %-----
0066 function ii = getInfo(varargin)
0067     if nargin == 1 && strcmpi(varargin{1}, 'None')
0068         sets = {};
0069         pl   = [];
0070     else
0071         sets = {'Default'};
0072         pl   = getDefaultPlist;
0073     end
0074     % Build info object
0075     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: mean.m,v 1.24
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0076 end
0077
0078 %-----
0079 % Get Default Plist
0080 %-----
0081 function pl_default = getDefaultPlist()
0082
0083     pl_default = plist('axis', 'xy');
0084 end
0085
0086 % END

```


median

PURPOSE ^

MEDIAN computes the median value of the data in the AO.

SYNOPSIS ^

function varargout = median(varargin)

DESCRIPTION ^

```
MEDIAN computes the median value of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MEDIAN computes the median value of the data in the AO.

CALL:      ao_out = median(ao_in);
           ao_out = median(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('median')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('median', 'None')

VERSION:    $Id: median.m,v 1.19 2008/09/05 11:15:19 ingo Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MEDIAN computes the median value of the data in the AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MEDIAN computes the median value of the data in the AO.
0005 %
0006 % CALL:      ao_out = median(ao_in);
0007 %           ao_out = median(ao_in, pl);
0008 %
0009 % PARAMETERS: see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 %              >> ao.getInfo('median')
```

```

0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> ao.getInfo('median', 'None')
0016 %
0017 % VERSION:      $Id: median.m,v 1.19 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:      12-03-07 M Hewitson
0020 %                Creation
0021 %
0022 %%%%%%%%%%%
0023
0024 function varargout = median(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Apply method to all AOs
0044 applymethod(bs, ao_invars, 'median', pl, getDefaultPlist, getInfo);
0045
0046 % remove fs and nsecs
0047 for j=1: numel(bs)
0048     if ismethod(bs(j).data, 'setFs')
0049         bs(j).setFs(NaN, 'internal');
0050     end
0051     if ismethod(bs(j).data, 'setNsecs')
0052         bs(j).data.setNsecs(NaN, 'internal');
0053     end
0054 end
0055
0056 % Set output
0057 if nargout > 0
0058     varargout{1} = bs;
0059 end
0060 end
0061
0062 %-----
0063 % Get Info Object
0064 %-----
0065 function ii = getInfo(varargin)
0066     if nargin == 1 && strcmpi(varargin{1}, 'None')
0067         sets = {};
0068         pl = [];
0069     else
0070         sets = {'Default'};
0071         pl = getDefaultPlist;
0072     end
0073     % Build info object
0074     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: median.m,v 1.19
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0075 end
0076
0077 %-----
0078 % Get Default Plist
0079 %-----
0080 function pl_default = getDefaultPlist()
0081
0082     pl_default = plist('axis', 'xy');
0083 end
0084
0085 % END

```

min

PURPOSE ^

MIN computes the minimum value of the data in the AO.

SYNOPSIS ^

function varargout = min(varargin)

DESCRIPTION ^

```
MIN computes the minimum value of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MIN computes the minimum value of the data in the AO.

CALL:      ao_out = min(ao_in);
           ao_out = min(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('min')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('min', 'None')

VERSION:    $Id: min.m,v 1.14 2008/09/08 08:31:56 hewitson Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [min](#) MIN computes the minimum value of the data in the AO.

This function is called by:

-  [attachm](#) ATTACHM attach an m file to the analysis object.
-  [display](#) DISPLAY implement terminal display for analysis object.
-  [fromDatafile](#) FROMDATAFILE Construct an ao from filename AND parameter list
-  [join](#) JOIN multiple AOs into a single AO.
-  [linedetect](#) LINEDETECT find spectral lines in the ao/fsdata objects.
-  [min](#) MIN computes the minimum value of the data in the AO.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl_default = getDefaultPlist()

SOURCE CODE ^

```

0001 % MIN computes the minimum value of the data in the A0.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MIN computes the minimum value of the data in the A0.
0005 %
0006 % CALL:          ao_out = min(ao_in);
0007 %              ao_out = min(ao_in, pl);
0008 %
0009 % PARAMETERS:    see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO:   Get information about this methods by calling
0012 %               >> ao.getInfo('min')
0013 %
0014 %               Get information about a specified set-plist by calling:
0015 %               >> ao.getInfo('min', 'None')
0016 %
0017 % VERSION:       $Id: min.m,v 1.14 2008/09/08 08:31:56 hewitson Exp $
0018 %
0019 % HISTORY:       12-03-07 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = min(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all A0s
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Get default parameters
0041 pl = combine(pl, getDefaultPlist);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargin);
0045
0046 % Apply method to all A0s
0047 for j=1: numel(bs)
0048     % get min value
0049     if isfield(bs(j).data, 'x')
0050         switch find(pl, 'axis')
0051             case 'x'
0052                 [mx, idx] = min(bs(j).data.getX);
0053                 my = bs(j).data.getY(idx);
0054             case 'y'
0055                 [my, idx] = min(bs(j).data.getY);
0056                 mx = bs(j).data.getX(idx);
0057         end
0058         bs(j).setXY(mx, my, 'internal');
0059     else
0060         [my, idx] = min(bs(j).data.getY);
0061         bs(j).setY(my, 'internal');
0062     end
0063     % remove fs and nsecs
0064     if ismethod(bs(j).data, 'setFs')
0065         bs(j).data.setFs(NaN, 'internal');
0066     end
0067     if ismethod(bs(j).data, 'setNsecs');
0068         bs(j).data.setNsecs(NaN, 'internal');
0069     end
0070     % append history
0071     bs(j).addHistory(getInfo('None'), pl, ao_invars(j), bs(j).hist);
0072     % Set new A0 name
0073     bs(j).setName(['min(' ao_invars{j} ')'], 'internal');
0074 end
0075
0076 % Set output
0077 if nargin > 0
0078     varargout{1} = bs;
0079 end
0080 end
0081
0082 %-----
0083 % Get Info Object
0084 %-----
0085 function ii = getInfo(varargin)
0086     if nargin == 1 && strcmpi(varargin{1}, 'None')

```

```

0087     sets = {};
0088     pl    = [];
0089     else
0090         sets = {'Default'};
0091         pl    = getDefaultPlist;
0092     end
0093     % Build info object
0094     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: min.m,v 1.14
2008/09/08 08:31:56 hewitson Exp $', sets, pl);
0095 end
0096
0097 %-----
0098 % Get Default Plist
0099 %-----
0100 function pl_default = getDefaultPlist()
0101
0102     pl_default = plist('axis', 'y');
0103 end
0104
0105 % END

```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

mode

PURPOSE ^

MODE computes the modal value of the data in the AO.

SYNOPSIS ^

function varargout = mode(varargin)

DESCRIPTION ^

MODE computes the modal value of the data in the AO.
%%%

DESCRIPTION: MODE computes the modal value of the data in the AO.

CALL: ao_out = mode(ao_in);
 ao_out = mode(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('mode')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('mode', 'None')

VERSION: \$Id: mode.m,v 1.9 2008/09/05 11:15:19 ingo Exp \$

HISTORY: 12-03-07 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MODE computes the modal value of the data in the AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MODE computes the modal value of the data in the AO.
0005 %
0006 % CALL:           ao_out = mode(ao_in);
0007 %           ao_out = mode(ao_in, pl);
0008 %
0009 % PARAMETERS: see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 %           >> ao.getInfo('mode')
```

```

0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> ao.getInfo('mode', 'None')
0016 %
0017 % VERSION:      $Id: mode.m,v 1.9 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:      12-03-07 M Hewitson
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%
0023
0024 function varargout = mode(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Apply method to all AOs
0044 applymethod(bs, ao_invars, 'mode', pl, getDefaultPlist, getInfo);
0045
0046 % remove fs and nsecs
0047 for j=1: numel(bs)
0048     if ismethod(bs(j).data, 'setFs')
0049         bs(j).setFs(NaN, 'internal');
0050     end
0051     if ismethod(bs(j).data, 'setNsecs')
0052         bs(j).data.setNsecs(NaN, 'internal');
0053     end
0054 end
0055
0056 % Set output
0057 if nargout > 0
0058     varargout{1} = bs;
0059 end
0060 end
0061
0062 %-----
0063 % Get Info Object
0064 %-----
0065 function ii = getInfo(varargin)
0066     if nargin == 1 && strcmpi(varargin{1}, 'None')
0067         sets = {};
0068         pl = [];
0069     else
0070         sets = {'Default'};
0071         pl = getDefaultPlist;
0072     end
0073     % Build info object
0074     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: mode.m,v 1.9
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0075 end
0076
0077 %-----
0078 % Get Default Plist
0079 %-----
0080 function pl_default = getDefaultPlist()
0081
0082     pl_default = plist('axis', 'xy');
0083 end
0084
0085 % END

```

norm

PURPOSE ^

NORM overloads the norm operator for Analysis Objects.

SYNOPSIS ^

function varargout = norm(varargin)

DESCRIPTION ^

```
NORM overloads the norm operator for Analysis Objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: NORM overloads the norm operator for Analysis Objects.

CALL:      b = norm (a,pl) % only with data = cdata
           b = norm (a)

INPUTS:    pl    - a parameter list
           a     - input analysis object

OUTPUTS:    like matlab fct

Parameters: 'option' - a string or value that can be submitted i.e. 'inf' for
                    the infinity norm of the input ao, the largest row sum
                    options are the same as for the matlab function

                    see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('norm')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('norm', 'None')

VERSION:    $Id: norm.m,v 1.22 2008/09/05 11:15:19 ingo Exp $

HISTORY:    08-05-07 A Monsky
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [welchscale](#) WELCHSCALE scales the output of welch to be in the required units

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```

0001 % NORM overloads the norm operator for Analysis Objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: NORM overloads the norm operator for Analysis Objects.
0005 %
0006 % CALL:      b = norm (a,pl) % only with data = cdata
0007 %           b = norm (a)
0008 %
0009 % INPUTS:    pl    - a parameter list
0010 %           a      - input analysis object
0011 %
0012 % OUTPUTS:    like matlab fct
0013 %
0014 % Parameters: 'option' - a string or value that can be submitted i.e. 'inf' for
0015 %               the infinity norm of the input ao, the largest row sum
0016 %               options are the same as for the matlab function
0017 %
0018 %               see help for data2D/applymethod for additional parameters
0019 %
0020 % M-FILE INFO: Get information about this methods by calling
0021 %               >> ao.getInfo('norm')
0022 %
0023 %               Get information about a specified set-plist by calling:
0024 %               >> ao.getInfo('norm', 'None')
0025 %
0026 % VERSION:    $Id: norm.m,v 1.22 2008/09/05 11:15:19 ingo Exp $
0027 %
0028 % HISTORY:    08-05-07 A Monsky
0029 %               Creation
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 function varargout = norm(varargin)
0034 %
0035 % Check if this is a call for parameters
0036 if utils.helper.isinfocall(varargin{:})
0037     varargout{1} = getInfo(varargin{3});
0038     return
0039 end
0040 %
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043 %
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047 %
0048 % Collect all AOs and plists
0049 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0051 %
0052 % Decide on a deep copy or a modify
0053 bs = copy(as, nargout);
0054 %
0055 % Combine plists
0056 pl = combine(pl, getDefaultPlist);
0057 %
0058 % go through analysis objects
0059 for j=1:numel(bs)
0060     switch class(bs(j).data)
0061     case 'cdata'
0062         % Apply norm method to data
0063         applymethod(bs(j), ao_invars(j), 'norm', pl, getDefaultPlist, getInfo);
0064     otherwise
0065         error('### this function works for cdata type AO only')
0066     end
0067 end
0068 %
0069 % Set output
0070 if nargout > 0
0071     varargout{1} = bs;
0072 end
0073 end
0074 %
0075 %-----
0076 % Get Info Object
0077 %-----
0078 function ii = getInfo(varargin)
0079 if nargin == 1 && strcmpi(varargin{1}, 'None')
0080     sets = {};
0081     pl = [];
0082 else
0083     sets = {'Default'};
0084     pl = getDefaultPlist;
0085 end

```

```
0086 % Build info object
0087 ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: norm.m,v 1.22
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0088 end
0089
0090 %-----
0091 % Get Default Plist
0092 %-----
0093 function pl = getDefaultPlist()
0094     pl = plist('option', '');
0095 end
0096 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

phase

PURPOSE ^

PHASE overloads the ltpda_phase operator for Analysis objects.

SYNOPSIS ^

function varargout = phase(varargin)

DESCRIPTION ^

PHASE overloads the ltpda_phase operator for Analysis objects.
%%%

DESCRIPTION: PHASE overloads the ltpda_phase operator for Analysis objects.
Uses utils.math.phase() to compute the phase angle.

CALL: ao_out = phase(ao_in);
 ao_out = phase(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('phase')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('phase', 'None')

VERSION: \$Id: phase.m,v 1.12 2008/09/05 11:15:19 ingo Exp \$

HISTORY: 22-08-2007 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [upsample](#) UPSAMPLE overloads upsample function for AOs.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % PHASE overloads the ltpda_phase operator for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PHASE overloads the ltpda_phase operator for Analysis objects.
0005 %               Uses utils.math.phase() to compute the phase angle.
0006 %
0007 % CALL:        ao_out = phase(ao_in);
0008 %             ao_out = phase(ao_in, pl);
```

```

0009 %
0010 % PARAMETERS: see help for data2D/applymethod for additional parameters
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 % >> ao.getInfo('phase')
0014 %
0015 % Get information about a specified set-plist by calling:
0016 % >> ao.getInfo('phase', 'None')
0017 %
0018 % VERSION: $Id: phase.m,v 1.12 2008/09/05 11:15:19 ingo Exp $
0019 %
0020 % HISTORY: 22-08-2007 M Hewitson
0021 % Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024 %
0025 function varargout = phase(varargin)
0026
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 import utils.const.*
0034 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0035
0036 % Collect input variable names
0037 in_names = cell(size(varargin));
0038 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0039
0040 % Collect all AOs
0041 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0043
0044 % Decide on a deep copy or a modify
0045 bs = copy(as, nargout);
0046
0047 % Combine plists
0048 pl = combine(pl, getDefaultPlist);
0049
0050 % Apply method to all AOs
0051 applymethod(bs, ao_invars, 'utils.math.phase', pl, getDefaultPlist, getInfo);
0052
0053 % Reshape the ouput to the same size of the input
0054 if nargout > 0
0055     varargout{1} = bs;
0056 end
0057 end
0058 %-----
0059 % Get Info Object
0060 %-----
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl = [];
0065     else
0066         sets = {'Default'};
0067         pl = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: phase.m,v 1.12
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0071 end
0072
0073 %-----
0074 % Get Default Plist
0075 %-----
0076 function pl = getDefaultPlist()
0077     pl = plist('axis', 'y');
0078 end
0079
0080 % END

```

real

PURPOSE ^

REAL overloads the real operator for Analysis objects.

SYNOPSIS ^

function varargout = real(varargin)

DESCRIPTION ^

```
REAL overloads the real operator for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: REAL overloads the real operator for Analysis objects.
CALL:      ao_out = real(ao_in);
           ao_out = real(ao_in, pl);
PARAMETERS: see help for data2D/applymethod for additional parameters
M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('real')
              Get information about a specified set-plist by calling:
              >> ao.getInfo('real', 'None')
VERSION:    $Id: real.m,v 1.11 2008/09/05 11:15:19 ingo Exp $
HISTORY:    20-0802007 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [export](#) EXPORT export an analysis object to a text file.
-  [iplot](#) IPLOT provides an intelligent plotting tool for LTPDA.
-  [plot](#) PLOT a simple plot of analysis objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % REAL overloads the real operator for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: REAL overloads the real operator for Analysis objects.
0005 %
0006 % CALL:      ao out = real(ao in);
```



```

0007 %          ao_out = real(ao_in, pl);
0008 %
0009 % PARAMETERS:  see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO:  Get information about this methods by calling
0012 %               >> ao.getInfo('real')
0013 %
0014 %               Get information about a specified set-plist by calling:
0015 %               >> ao.getInfo('real', 'None')
0016 %
0017 % VERSION:      $Id: real.m,v 1.11 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:      20-0802007 Diepholz
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = real(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 import utils.const.*
0033 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs and plists
0040 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0041 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargout);
0045
0046 % Combine plists
0047 pl = combine(pl, getDefaultPlist);
0048
0049 % Apply method to all AOs
0050 applymethod(bs, ao_invars, 'real', pl, getDefaultPlist, getInfo);
0051
0052 % Reshape the ouput to the same size of the input
0053 if nargout > 0
0054     varargout{1} = bs;
0055 end
0056 end
0057
0058 %-----
0059 % Get Info Object
0060 %-----
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl    = [];
0065     else
0066         sets = {'Default'};
0067         pl    = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: real.m,v 1.11
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0071 end
0072
0073 %-----
0074 % Get Default Plist
0075 %-----
0076 function pl_default = getDefaultPlist()
0077     pl_default = plist('axis', 'y');
0078 end
0079
0080 % END

```

sign

PURPOSE ^

SIGN overloads the sign operator for Analysis objects.%

SYNOPSIS ^

function varargout = sign (varargin)

DESCRIPTION ^

```
SIGN overloads the sign operator for Analysis objects.%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SIGN overloads the sign operator for Analysis objects.

CALL:
    ao_out = sign(ao_in);
    ao_out = sign(ao_in, pl);
    ao_out = sign(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('sign')

    Get information about a specified set-plist by calling:
    >> ao.getInfo('sign', 'None')

VERSION:    $Id: sign.m,v 1.7 2008/09/05 11:15:19 ingo Exp $

HISTORY:    23-05-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SIGN overloads the sign operator for Analysis objects.%
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SIGN overloads the sign operator for Analysis objects.
0005 %
0006 % CALL:
0007 %     ao_out = sign(ao_in);
0008 %     ao_out = sign(ao_in, pl);
0009 %     ao_out = sign(ao1, pl1, ao_vector, ao_matrix, pl2);
0010 % PARAMETERS: see help for data2D/applymethod for additional parameters
0011 %
```

```

0012 % M-FILE INFO: Get information about this methods by calling
0013 %                >> ao.getInfo('sign')
0014 %
0015 %                Get information about a specified set-plist by calling:
0016 %                >> ao.getInfo('sign', 'None')
0017 %
0018 % VERSION:      $Id: sign.m,v 1.7 2008/09/05 11:15:19 ingo Exp $
0019 %
0020 % HISTORY:      23-05-2007 Diepholz
0021 %                Creation
0022 %
0023 %%%%%%%%%%%
0024 %%%%%%%%%%%
0025 function varargout = sign (varargin)
0026 %
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032 %
0033 import utils.const.*
0034 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0035 %
0036 % Collect input variable names
0037 in_names = cell(size(varargin));
0038 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0039 %
0040 % Collect all AOs
0041 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0043 %
0044 % Decide on a deep copy or a modify
0045 bs = copy(as, nargsout);
0046 %
0047 % Combine plists
0048 pl = combine(pl, getDefaultPlist);
0049 %
0050 % Apply method to all AOs
0051 applymethod(bs, ao_invars, 'sign', pl, getDefaultPlist, getInfo);
0052 %
0053 % Reshape the ouput to the same size of the input
0054 if nargsout > 0
0055     varargout{1} = bs;
0056 end
0057 end
0058 %
0059 %-----
0060 % Get Info Object
0061 %-----
0062 function ii = getInfo(varargin)
0063     if nargin == 1 && strcmpi(varargin{1}, 'None')
0064         sets = {};
0065         pl   = [];
0066     else
0067         sets = {'Default'};
0068         pl   = getDefaultPlist;
0069     end
0070     % Build info object
0071     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: sign.m,v 1.7
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0072 end
0073 %
0074 %-----
0075 % Get Default Plist
0076 %-----
0077 function pl_default = getDefaultPlist()
0078     pl_default = plist('axis', 'y');
0079 end
0080 % END
0081

```

sort

PURPOSE ^

SORT the values in the AO.

SYNOPSIS ^

function varargout = **sort** (varargin)

DESCRIPTION ^

```
SORT the values in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SORT sorts the values in the input data.

CALL: ao_out = sort(ao_in);
      ao_out = sort(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS:

    'dim' - sort along the specified dimension: 'x' or 'y'
           [default: 'y']
    'dir' - select direction of sort: 'ascend' or 'descend'
           [default: 'ascend']

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('sort')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('sort', 'None')

VERSION: $Id: sort.m,v 1.10 2008/09/05 11:15:19 ingo Exp $

HISTORY: 27-05-2008 Hewitson
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [sort](#) SORT the values in the AO.

This function is called by:

-  [fromDatafile](#) FROMDATAFILE Construct an ao from filename AND parameter list
-  [join](#) JOIN multiple AOs into a single AO.
-  [linedetect](#) LINEDETECT find spectral lines in the ao/fsdata objects.
-  [smoother](#) SMOOTHER smooths a given series of data points using the specified method.
-  [sort](#) SORT the values in the AO.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % SORT the values in the AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SORT sorts the values in the input data.
0005 %
0006 % CALL: ao_out = sort(ao_in);
0007 %         ao_out = sort(ao1, pl1, ao_vector, ao_matrix, pl2);
0008 %
0009 % PARAMETERS:
0010 %
0011 %         'dim' - sort along the specified dimension: 'x' or 'y'
0012 %                  [default: 'y']
0013 %         'dir' - select direction of sort: 'ascend' or 'descend'
0014 %                  [default: 'ascend']
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 %               >> ao.getInfo('sort')
0018 %
0019 %               Get information about a specified set-plist by calling:
0020 %               >> ao.getInfo('sort', 'None')
0021 %
0022 % VERSION: $Id: sort.m,v 1.10 2008/09/05 11:15:19 ingo Exp $
0023 %
0024 % HISTORY: 27-05-2008 Hewitson
0025 %           Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = sort (varargin)
0030
0031 % Check if this is a call for parameters
0032 if utils.helper.isinfocall(varargin{:})
0033     varargout{1} = getInfo(varargin{3});
0034     return
0035 end
0036
0037 import utils.const.*
0038 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0039
0040 % Collect input variable names
0041 in_names = cell(size(varargin));
0042 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0043
0044 % Collect all AOs
0045 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0046 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0047
0048 % Decide on a deep copy or a modify
0049 bs = copy(as, nargout);
0050
0051 % Combine plists
0052 pl = combine(pl, getDefaultPlist);
0053
0054 % Get parameters
0055 sdir = find(pl, 'dir');
0056 dim = find(pl, 'dim');
0057
0058 %% go through analysis objects
0059 for j=1:numel(bs)
0060     % Sort values
0061     switch dim
0062     case 'x'
0063         [mx, idx] = sort(bs(j).data.getX, 1, sdir);
0064         my = bs(j).data.y(idx);
0065     case 'y'
0066         [my, idx] = sort(bs(j).data.y, 1, sdir);
0067         mx = bs(j).data.getX(idx);
0068     otherwise
0069         error('### can''t sort along dimension ''%s''.', dim);
0070     end
0071     % set new data
0072     bs(j).setXY(mx, my, 'internal');
0073     % set data name
0074     bs(j).setName(sprintf('sort(%s)', ao_invars{j}), 'internal');
0075     % Add history
0076     bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0077 end
0078
0079 % Reshape the ouput to the same size of the input
0080 if nargout > 0
0081     varargout{1} = bs;
0082 end

```

```
0083 end
0084
0085 %-----
0086 % Get Info Object
0087 %-----
0088 function ii = getInfo(varargin)
0089     if nargin == 1 && strcmpi(varargin{1}, 'None')
0090         sets = {};
0091         pl = [];
0092     else
0093         sets = {'Default'};
0094         pl = getDefaultPlist;
0095     end
0096     % Build info object
0097     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: sort.m,v 1.10
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0098 end
0099
0100 %-----
0101 % Get Default Plist
0102 %-----
0103 function pl_default = getDefaultPlist()
0104     pl_default = plist('dim', 'y', 'dir', 'ascend');
0105 end
0106
0107 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

sqrt

PURPOSE ^

SQRT computes the square root of the data in the AO.

SYNOPSIS ^

function varargout = sqrt(varargin)

DESCRIPTION ^

```
SQRT computes the square root of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SQRT computes the square root of the data in the AO.

CALL:      ao_out = sqrt(ao_in);
           ao_out = sqrt(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('sqrt')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('sqrt', 'None')

VERSION:    $Id: sqrt.m,v 1.24 2008/09/05 11:15:19 ingo Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [conv_noisegen](#) CONV_NOISEGEN calls the matlab function conv.m to convolute poles and zeros from a given pzmodel
-  [diff](#) DIFF differentiates the data in AO.
-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input PSD.
-  [lpsd](#) LPSD implement the LPSD algorithm for analysis objects.
-  [ltf_plan](#) LTF_PLAN computes all input values needed for the LPSD and LTFE algorithms.
-  [mchol](#) MCHOL
-  [mdc1_ifo2acc_fd](#) MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
-  [mdc1_x2acc](#) MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter
-  [rms](#) RMS Calculate RMS deviation from spectrum
-  [welchscale](#) WELCHSCALE scales the output of welch to be in the required units

SUBFUNCTIONS

 `function ii = getInfo(varargin)`
 `function pl_default = getDefaultPlist()`

SOURCE CODE ^

```

0001 % Sqrt computes the square root of the data in the A0.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Sqrt computes the square root of the data in the A0.
0005 %
0006 % CALL:          ao_out = sqrt(ao_in);
0007 %               ao_out = sqrt(ao_in, pl);
0008 %
0009 % PARAMETERS:    see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO:   Get information about this methods by calling
0012 %               >> ao.getInfo('sqrt')
0013 %
0014 %               Get information about a specified set-plist by calling:
0015 %               >> ao.getInfo('sqrt', 'None')
0016 %
0017 % VERSION:       $Id: sqrt.m,v 1.24 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:       12-03-07 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = sqrt(varargin)
0025
0026     import utils.const.*
0027
0028     % Check if this is a call for parameters
0029     if utils.helper.isinfocall(varargin{:})
0030         varargout{1} = getInfo(varargin{3});
0031         return
0032     end
0033
0034     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0035
0036     % Collect input variable names
0037     in_names = cell(size(varargin));
0038     for ii = 1:nargin, in_names{ii} = inputname(ii); end
0039
0040     % Collect all A0s
0041     [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042     pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0043
0044     % Get default parameters
0045     pl = combine(pl, getDefaultPlist);
0046
0047     % Decide on a deep copy or a modify
0048     bs = copy(as, nargout);
0049
0050     % Apply method to all A0s
0051     applymethod(bs, ao_invars, 'sqrt', pl, getDefaultPlist, getInfo);
0052
0053     % Set units
0054     for ii = 1:numel(bs)
0055         app_axis = pl.find('axis');
0056         if ~isempty(find('X'==upper(app_axis), 1))
0057             bs(ii).setXunits(feval('sqrt', bs(ii).data.xunits), 'internal');
0058         end
0059         if ~isempty(find('Y'==upper(app_axis), 1))
0060             bs(ii).setYunits(feval('sqrt', bs(ii).data.yunits), 'internal');
0061         end
0062     end
0063
0064     % Set output
0065     if nargout > 0
0066         varargout{1} = bs;
0067     end
0068 end
0069
0070 %-----
0071 % Get Info Object
0072 %-----
0073 function ii = getInfo(varargin)
0074     if nargin == 1 && strcmpi(varargin{1}, 'None')
```



```
0075     sets = {};
0076     pl    = [];
0077     else
0078         sets = {'Default'};
0079         pl    = getDefaultPlist;
0080     end
0081     % Build info object
0082     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: sqrt.m,v 1.24
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0083 end
0084
0085 %-----
0086 % Get Default Plist
0087 %-----
0088 function pl_default = getDefaultPlist()
0089
0090     pl_default = plist('axis', 'y');
0091 end
0092
0093 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

std

PURPOSE ^

STD computes the standard deviation of the data in the AO.

SYNOPSIS ^

function varargout = std(varargin)

DESCRIPTION ^

```
STD computes the standard deviation of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: STD computes the standard deviation of the data in the AO.
CALL: ao_out = std(ao_in);
      ao_out = std(ao_in, pl);
PARAMETERS: see help for data2D/applymethod for additional parameters
M-FILE INFO: Get information about this methods by calling
              >>
              Get information about a specified set-plist by calling:
              >> ao.getInfo('std', 'None')
VERSION: $Id: std.m,v 1.18 2008/09/05 11:15:19 ingo Exp $
HISTORY: 12-03-07 M Hewitson
          Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [gapfilling](#) GAPFILLING fills possible gaps in data.
-  [spikecleaning](#) spikecleaning detects and corrects possible spikes in analysis objects

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STD computes the standard deviation of the data in the AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STD computes the standard deviation of the data in the AO.
0005 %
0006 % CALL: ao_out = std(ao_in);
0007 %         ao_out = std(ao_in, pl);
```

```

0008 %
0009 % PARAMETERS: see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 % >>
0013 %
0014 % Get information about a specified set-plist by calling:
0015 % >> ao.getInfo('std', 'None')
0016 %
0017 % VERSION: $Id: std.m,v 1.18 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY: 12-03-07 M Hewitson
0020 % Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = std(varargin)
0025 %
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035 %
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039 %
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042 %
0043 % Apply method to all AOs
0044 applymethod(bs, ao_invars, 'std', pl, getDefaultPlist, getInfo);
0045 %
0046 % remove fs and nsecs
0047 for j=1: numel(bs)
0048     if ismethod(bs(j).data, 'setFs')
0049         bs(j).setFs(NaN, 'internal');
0050     end
0051     if ismethod(bs(j).data, 'setNsecs')
0052         bs(j).data.setNsecs(NaN, 'internal');
0053     end
0054 end
0055 %
0056 % Set output
0057 if nargout > 0
0058     varargout{1} = bs;
0059 end
0060 end
0061 %-----
0062 % Get Info Object
0063 %-----
0064 %
0065 function ii = getInfo(varargin)
0066 if nargin == 1 && strcmpi(varargin{1}, 'None')
0067     sets = {};
0068     pl = [];
0069 else
0070     sets = {'Default'};
0071     pl = getDefaultPlist;
0072 end
0073 % Build info object
0074 ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: std.m,v 1.18
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0075 end
0076 %-----
0077 % Get Default Plist
0078 %-----
0079 %
0080 function pl_default = getDefaultPlist()
0081 pl_default = plist('axis', 'y');
0082 end
0083 %
0084 %
0085 % END

```

sum

PURPOSE ^

SUM computes the sum of the data in the AO.

SYNOPSIS ^

function varargout = sum(varargin)

DESCRIPTION ^

```
SUM computes the sum of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SUM computes the sum of the data in the AO.

CALL:      ao_out = sum(ao_in);
           ao_out = sum(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('sum')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('sum', 'None')

VERSION:    $Id: sum.m,v 1.15 2008/09/05 11:15:19 ingo Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [computeperiodogram](#) COMPUTEPERIODOGRAM Periodogram spectral estimation.
-  [diff](#) DIFF differentiates the data in AO.
-  [fromDatafile](#) FROMDATAFILE Construct an ao from filename AND parameter list
-  [fromTSfcn](#) FROMTSFCN Construct an ao from a ts-function string
-  [validate](#) VALIDATE checks that the input Analysis Object is reproducible and valid.
-  [welchscale](#) WELCHSCALE scales the output of welch to be in the required units

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SUM computes the sum of the data in the AO.
```

```

0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SUM computes the sum of the data in the A0.
0005 %
0006 % CALL:          ao_out = sum(ao_in);
0007 %               ao_out = sum(ao_in, pl);
0008 %
0009 % PARAMETERS:    see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO:   Get information about this methods by calling
0012 %               >> ao.getInfo('sum')
0013 %
0014 %               Get information about a specified set-plist by calling:
0015 %               >> ao.getInfo('sum', 'None')
0016 %
0017 % VERSION:       $Id: sum.m,v 1.15 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:       12-03-07 M Hewitson
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = sum(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Apply method to all AOs
0044 applymethod(bs, ao_invars, 'sum', pl, getDefaultPlist, getInfo);
0045
0046 % Set output
0047 if nargout > 0
0048     varargout{1} = bs;
0049 end
0050 end
0051
0052 %-----
0053 % Get Info Object
0054 %-----
0055 function ii = getInfo(varargin)
0056     if nargin == 1 && strcmpi(varargin{1}, 'None')
0057         sets = {};
0058         pl    = [];
0059     else
0060         sets = {'Default'};
0061         pl    = getDefaultPlist;
0062     end
0063     % Build info object
0064     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: sum.m,v 1.15
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0065 end
0066
0067 %-----
0068 % Get Default Plist
0069 %-----
0070 function pl_default = getDefaultPlist()
0071
0072     pl_default = plist('axis', 'y');
0073 end
0074
0075 % END

```

svd

PURPOSE ^

SVD overloads the determinant function for Analysis objects.

SYNOPSIS ^

function varargout = svd(varargin)

DESCRIPTION ^

```
SVD overloads the determinant function for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: SVD overloads the determinant function for Analysis objects.
CALL:      u      = svd (a,pl)
INPUTS:    a      - input analysis object
            pl     - a parameter list
OUTPUTS:    like matlab fct
PARAMETERS: 'option' - a string or value that can be submitted i.e. 'econ'
              to produce economy size decomposition
              - options are the same as for the matlab function

    see help for data2D/applymethod for additional parameters
The following call returns an info object for this method.
>> info = ao.getInfo('svd')
VERSION:    $Id: svd.m,v 1.19 2008/09/05 11:15:19 ingo Exp $
HISTORY:    08-05-07A Monsky
              Creation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SVD overloads the determinant function for Analysis objects.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: SVD overloads the determinant function for Analysis objects.
```

```

0006 %
0007 % CALL:      u      = svd (a,pl)
0008 %
0009 % INPUTS:    a      - input analysis object
0010 %            pl     - a parameter list
0011 %
0012 % OUTPUTS:    like matlab fct
0013 %
0014 % PARAMETERS: 'option' - a string or value that can be submitted i.e. 'econ'
0015 %              to produce economy size decomposition
0016 %              - options are the same as for the matlab function
0017 %
0018 %      see help for data2D/applymethod for additional parameters
0019 %
0020 % The following call returns an info object for this method.
0021 %
0022 % >> info = ao.getInfo('svd')
0023 %
0024 % VERSION:    $Id: svd.m,v 1.19 2008/09/05 11:15:19 ingo Exp $
0025 %
0026 % HISTORY:    08-05-07A Monsky
0027 %              Creation.
0028 %
0029 %%%%%%%%%%%
0030 %
0031 %
0032 % FOR THESE OTHER OUTPUT OPTIONS WE CAN'T USE THE GENERIC APPLYMETHOD. IF
0033 % WE NEED THIS FUNCTIONALITY THEN WE SHOULD RECODE THIS INDIVIDUALLY.
0034 %
0035 %      [u, s] = svd (a,pl)
0036 %      [u, s, v] = svd (a,pl)
0037 %
0038 %
0039 %
0040 function varargout = svd(varargin)
0041 %
0042 %% Check if this is a call for parameters
0043 if utils.helper.isinfocall(varargin{:})
0044     varargout{1} = getInfo(varargin{3});
0045     return
0046 end
0047 %
0048 % Collect input variable names
0049 in_names = cell(size(varargin));
0050 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0051 %
0052 % Collect all AOs
0053 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0054 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0055 %
0056 % Decide on a deep copy or a modify
0057 bs = copy(as, nargin);
0058 %
0059 % Combine plists
0060 pl = combine(pl, getDefaultPlist);
0061 %
0062 %% go through analysis objects
0063 for j=1:numel(bs)
0064     if ~isa(bs(j).data, 'cdata')
0065         warning('!!! This methods works for cdata type AO only. Skipping AO %s',
0066             ao_invars{j})
0067     else
0068         % Apply method to all AOs
0069         applymethod(bs(j), ao_invars(j), 'svd', pl, getDefaultPlist, getInfo);
0070     end
0071 end
0072 %
0073 % Set output
0074 if nargin > 0
0075     varargout{1} = bs;
0076 end
0077 %
0078 %-----
0079 % Get Info Object
0080 %-----
0081 function ii = getInfo(varargin)
0082 if nargin == 1 && strcmpi(varargin{1}, 'None')
0083     sets = {};
0084     pl = [];
0085 else
0086     sets = {'Default'};
0087     pl = getDefaultPlist;
0088 end
0089 % Build info object
0090 ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: svd.m,v 1.19

```

```
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0091 end
0092 %-----
0093 % Get Default Plist
0094 %-----
0095 function pl = getDefaultPlist()
0096     pl = plist('option', '');
0097 end
0098
0099 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

transpose

PURPOSE ^

TRANPOSE overloads the .' operator for Analysis Objects.

SYNOPSIS ^

function varargout = transpose(varargin)

DESCRIPTION ^

```
TRANPOSE overloads the .' operator for Analysis Objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: TRANPOSE overloads the .' operator for Analysis Objects.

CALL:      a = a1.'

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('transpose')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('transpose', 'None')

VERSION:    $Id: transpose.m,v 1.22 2008/09/05 11:15:19 ingo Exp $

HISTORY:    08-05-07 A Monsky
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % TRANPOSE overloads the .' operator for Analysis Objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TRANPOSE overloads the .' operator for Analysis Objects.
0005 %
0006 % CALL:      a = a1.'
0007 %
0008 % PARAMETERS: see help for data2D/applymethod for additional parameters
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
```

```

0011 %                >> ao.getInfo('transpose')
0012 %
0013 %                Get information about a specified set-plist by calling:
0014 %                >> ao.getInfo('transpose', 'None')
0015 %
0016 % VERSION:      $Id: transpose.m,v 1.22 2008/09/05 11:15:19 ingo Exp $
0017 %
0018 % HISTORY:      08-05-07 A Monsky
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%
0022 %
0023 function varargout = transpose(varargin)
0024 %
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033 %
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037 %
0038 % Collect all AOs
0039 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0041 %
0042 % Decide on a deep copy or a modify
0043 bs = copy(as, nargout);
0044 %
0045 % Combine plists
0046 pl = combine(pl, getDefaultPlist);
0047 %
0048 axis = find(pl, 'axis');
0049 %
0050 for kk = 1:numel(bs)
0051     % Special case for cdata-objects because this objects doesn't have x values
0052     if isa(bs(kk).data, 'data2D')
0053         x = bs(kk).data.x;
0054         y = bs(kk).data.y;
0055         % Set the y-value to an empty array because the y-data keeps always the same
0056         % shape
0057         bs(kk).data.setY([]);
0058         bs(kk).data.setX([]);
0059         switch upper(axis)
0060             case 'X'
0061                 bs(kk).data.setX(x.);
0062                 bs(kk).data.setY(y.);
0063             case 'Y'
0064                 bs(kk).data.setX(x.);
0065                 bs(kk).data.setY(y.);
0066             case 'XY'
0067                 bs(kk).data.setX(x.);
0068                 bs(kk).data.setY(y.);
0069             otherwise
0070                 error('### Unknown axis to operate on.');

```

```
0097 %-----
0098 function ii = getInfo(varargin)
0099     if nargin == 1 && strcmpi(varargin{1}, 'None')
0100         sets = {};
0101         pl    = [];
0102     else
0103         sets = {'Default'};
0104         pl    = getDefaultPlist;
0105     end
0106     % Build info object
0107     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: transpose.m,v 1.22
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0108 end
0109
0110 %-----
0111 % Get Default Plist
0112 %-----
0113 function pl_default = getDefaultPlist()
0114     pl_default = plist('axis', 'xy');
0115 end
0116
0117 % END
```

uminus

PURPOSE ^

UMINUS overloads the uminus operator for Analysis objects.

SYNOPSIS ^

function varargout = uminus (varargin)

DESCRIPTION ^

```
UMINUS overloads the uminus operator for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: UMINUS overloads the uminus operator for Analysis objects.
             uminus(ao) negates all the elements of ao.data.

CALL:
    ao_out = uminus(ao_in);
    ao_out = uminus(ao_in, pl);
    ao_out = uminus(ao1, pl1, ao_vector, ao_matrix, pl2);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('times')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('times', 'None')

VERSION:     $Id: uminus.m,v 1.7 2008/09/05 11:15:19 ingo Exp $

HISTORY:     06-05-2008 Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % UMINUS overloads the uminus operator for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: UMINUS overloads the uminus operator for Analysis objects.
0005 %             uminus(ao) negates all the elements of ao.data.
0006 %
0007 % CALL:
0008 %     ao_out = uminus(ao_in);
0009 %     ao_out = uminus(ao_in, pl);
0010 %     ao_out = uminus(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 % M-FILE INFO: Get information about this methods by calling
0012 %             >> ao.getInfo('times')
```

```

0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> ao.getInfo('times', 'None')
0016 %
0017 % VERSION:      $Id: uminus.m,v 1.7 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:      06-05-2008 Hewitson
0020 %                Creation
0021 %
0022 %%%%%%%%%%%
0023
0024 function varargout = uminus (varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 import utils.const.*
0033 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs
0040 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0041 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargout);
0045
0046 % Combine plists
0047 pl = combine(pl, getDefaultPlist);
0048
0049 % Apply method to all AOs
0050 applymethod(bs, ao_invars, 'uminus', pl, getDefaultPlist, getInfo);
0051
0052 % Set output
0053 if nargout > 0
0054     varargout{1} = bs;
0055 end
0056 end
0057
0058 %-----
0059 % Get Info Object
0060 %-----
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl = [];
0065     else
0066         sets = {'Default'};
0067         pl = getDefaultPlist;
0068     end
0069 % Build info object
0070 ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: uminus.m,v 1.7
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0071 end
0072
0073 %-----
0074 % Get Default Plist
0075 %-----
0076 function pl_default = getDefaultPlist()
0077     pl_default = plist('axis', 'y');
0078 end
0079 % END

```

var

PURPOSE ^

VAR computes the variance of the data in the AO.

SYNOPSIS ^

function varargout = var(varargin)

DESCRIPTION ^

```
VAR computes the variance of the data in the AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: VAR computes the variance of the data in the AO.

CALL:      ao_out = var(ao_in);
           ao_out = var(ao_in, pl);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('var')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('var', 'None')

VERSION:    $Id: var.m,v 1.18 2008/09/05 11:15:19 ingo Exp $

HISTORY:    12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % VAR computes the variance of the data in the AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: VAR computes the variance of the data in the AO.
0005 %
0006 % CALL:      ao_out = var(ao_in);
0007 %           ao_out = var(ao_in, pl);
0008 %
0009 % PARAMETERS: see help for data2D/applymethod for additional parameters
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 %              >> ao.getInfo('var')
```

```

0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> ao.getInfo('var', 'None')
0016 %
0017 % VERSION:      $Id: var.m,v 1.18 2008/09/05 11:15:19 ingo Exp $
0018 %
0019 % HISTORY:      12-03-07 M Hewitson
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%
0023
0024 function varargout = var(varargin)
0025
0026 % Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Apply method to all AOs
0044 applymethod(bs, ao_invars, 'var', pl, getDefaultPlist, getInfo);
0045
0046 % remove fs and nsecs
0047 for j=1: numel(bs)
0048     if ismethod(bs(j).data, 'setFs')
0049         bs(j).setFs(NaN, 'internal');
0050     end
0051     if ismethod(bs(j).data, 'setNsecs')
0052         bs(j).data.setNsecs(NaN, 'internal');
0053     end
0054 end
0055
0056 % Set output
0057 if nargout > 0
0058     varargout{1} = bs;
0059 end
0060 end
0061
0062 %-----
0063 % Get Info Object
0064 %-----
0065 function ii = getInfo(varargin)
0066     if nargin == 1 && strcmpi(varargin{1}, 'None')
0067         sets = {};
0068         pl = [];
0069     else
0070         sets = {'Default'};
0071         pl = getDefaultPlist;
0072     end
0073     % Build info object
0074     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: var.m,v 1.18
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0075 end
0076
0077 %-----
0078 % Get Default Plist
0079 %-----
0080 function pl_default = getDefaultPlist()
0081     pl_default = plist('axis', 'xy');
0082 end
0083
0084 % END

```

zeropad

PURPOSE ^

ZEROPAD zero pads the input data series.

SYNOPSIS ^

function varargout = zeropad(varargin)

DESCRIPTION ^

```
ZEROPAD zero pads the input data series.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: ZEROPAD zero pads the input data series.

CALL:      b = zeropad(a, pl)

PARAMETERS: 'Factor'   - Pad to <Factor> times the input data length
               [default: 2]
            or
            'N'        - Pad with N zero samples

M-FILE INFO: Get information about this methods by calling
               >> ao.getInfo('zeropad')

               Get information about a specified set-plist by calling:
               >> ao.getInfo('zeropad', 'None')

VERSION:     $Id: zeropad.m,v 1.11 2008/09/05 11:15:19 ingo Exp $

HISTORY:     25-04-08 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY makes a (deep) copy of the input AOs.
-  **find** FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

SUBFUNCTIONS ^

-  **function ii = getInfo(varargin)**
-  **function pl = getDefaultPlist()**

SOURCE CODE ^

```
0001 % ZEROPAD zero pads the input data series.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ZEROPAD zero pads the input data series.
0005 %
0006 % CALL:      b = zeropad(a, pl)
0007 %
0008 % PARAMETERS: 'Factor'   - Pad to <Factor> times the input data length
0009 %               [default: 2]
0010 %            or
```



```

0011 %           'N'           - Pad with N zero samples
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %           >> ao.getInfo('zeropad')
0015 %
0016 %           Get information about a specified set-plist by calling:
0017 %           >> ao.getInfo('zeropad', 'None')
0018 %
0019 % VERSION:      $Id: zeropad.m,v 1.11 2008/09/05 11:15:19 ingo Exp $
0020 %
0021 % HISTORY:      25-04-08 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 function varargout = zeropad(varargin)
0027
0028 % Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 import utils.const.*
0035 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0036
0037 % Collect input variable names
0038 in_names = cell(size(varargin));
0039 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0040
0041 % Collect all AOs and plists
0042 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0043 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0044
0045 %% Decide on a deep copy or a modify
0046 bs = copy(as, nargin);
0047
0048 % combine plists
0049 pl = combine(pl, getDefaultPlist());
0050
0051 % Extract necessary parameters
0052 Factor = find(pl, 'Factor');
0053 N       = find(pl, 'N');
0054
0055 if isempty(Factor) && isempty(N)
0056     error('### Specify either a padding Factor, or a number of samples.');
```

```

0057 end
0058
0059 % Loop over input AOs
0060 for j=1:length(bs)
0061     if ~isa(bs(j).data, 'tsdata')
0062         warning('!!! Zero padding only works on time-series. Skipping AO %s',
ao_invars{j});
0063         bs(j) = [];
0064     else
0065         % Check the time-series is evenly samples
0066         if ~isempty(bs(j).data.x)
0067             error('### Zero padding only makes sense on evenly sampled time-series.
Resample first.');
```

```

0068         end
0069         if isempty(N)
0070             if Factor < 2
0071                 error('### Padding factor must be >= 2')
0072             end
0073             bs(j).setY([bs(j).data.getY; repmat(zeros(size(bs(j).data.getY)), Factor-1, 1)],
'internal');
0074         else
0075             if N > 0
0076                 bs(j).setY([bs(j).data.getY; zeros(N,1)], 'internal');
0077             else
0078                 error('### Number of samples to pad must be > 0');
0079             end
0080         end
0081         % Set name
0082         bs(j).setName(sprintf('zeropad(%s)', ao_invars{j}), 'internal');
0083         % Correct Nsecs
0084         bs(j).data.fixNsecs;
0085         % Add history
0086         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0087     end
0088 end
0089
0090 % Set outputs
0091 if nargin > 0
0092     varargout{1} = bs;
0093 end

```

```
0094 end
0095
0096 %-----
0097 % Get Info Object
0098 %-----
0099 function ii = getInfo(varargin)
0100     if nargin == 1 && strcmpi(varargin{1}, 'None')
0101         sets = {};
0102         pl    = [];
0103     else
0104         sets = {'Default'};
0105         pl    = getDefaultPlist;
0106     end
0107     % Build info object
0108     ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: zeropad.m,v 1.11
2008/09/05 11:15:19 ingo Exp $', sets, pl);
0109 end
0110
0111 %-----
0112 % Get Default Plist
0113 %-----
0114 function pl = getDefaultPlist()
0115     pl = plist('Factor', 2);
0116 end
0117 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

char

PURPOSE ^

CHAR overloads char() function for analysis objects.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR overloads char() function for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR overloads char() function for analysis objects.

CALL:      c = char(ao_in);

FORMAT:    c = 'ao.name/ao.data.name class(ao.data) [size(ao.data.yaxis)]'

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('char')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('char', 'None')

VERSION:   $Id: char.m,v 1.18 2008/09/05 11:05:28 ingo Exp $

HISTORY:   15-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR overloads char() function for analysis objects.

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [char](#) CHAR overloads char() function for analysis objects.
-  [display](#) DISPLAY implement terminal display for analysis object.
-  [iplot](#) IPLOT provides an intelligent plotting tool for LTPDA.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR overloads char() function for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR overloads char() function for analysis objects.
0005 %
0006 % CALL:      c = char(ao_in);
0007 %
```

```

0008 % FORMAT:      c = 'ao.name/ao.data.name class(ao.data) [size(ao.data.yaxis)]'
0009 %
0010 % M-FILE INFO:  Get information about this methods by calling
0011 %               >> ao.getInfo('char')
0012 %
0013 %               Get information about a specified set-plist by calling:
0014 %               >> ao.getInfo('char', 'None')
0015 %
0016 % VERSION:      $Id: char.m,v 1.18 2008/09/05 11:05:28 ingo Exp $
0017 %
0018 % HISTORY:      15-02-07 M Hewitson
0019 %               Creation
0020 %
0021 %%%%%%%%%%%%%%
0022 %
0023 function varargout = char(varargin)
0024
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0037
0038 % Collect all AOs and plists
0039 as = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040
0041 % go through analysis objects
0042 pstr = [];
0043 for j=1: numel(as)
0044     pstr = [pstr as(j).name];
0045     % get data type
0046     if isempty(as(j).data)
0047         pstr = [pstr ' / No data-object'];
0048     else
0049         pstr = [pstr '/' class(as(j).data)];
0050         pstr = [pstr char(as(j).data) ', '];
0051     end
0052 end
0053
0054 pstr = strrep(pstr, '_', '\_');
0055 varargout{1} = pstr(1:end-2);
0056
0057 end
0058
0059 %-----
0060 % Get Info Object
0061 %-----
0062 function ii = getInfo(varargin)
0063     if nargin == 1 && strcmpi(varargin{1}, 'None')
0064         sets = {};
0065         pl = [];
0066     else
0067         sets = {'Default'};
0068         pl = getDefaultPlist;
0069     end
0070     % Build info object
0071     ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: char.m,v 1.18
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0072 end
0073
0074 %-----
0075 % Get Default Plist
0076 %-----
0077 function pl_default = getDefaultPlist()
0078     pl_default = plist();
0079 end
0080

```

display

PURPOSE ^

DISPLAY implement terminal display for analysis object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY implement terminal display for analysis object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY implement terminal display for analysis object.

CALL:      txt = display(ao)
           ao                                % without a semicolon

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('display')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('display', 'None')

VERSION:    $Id: display.m,v 1.30 2008/09/05 11:05:29 ingo Exp $

HISTORY:    31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR overloads char() function for analysis objects.
-  [display](#) DISPLAY implement terminal display for analysis object.
-  [min](#) MIN computes the minimum value of the data in the AO.

This function is called by:

-  [display](#) DISPLAY implement terminal display for analysis object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY implement terminal display for analysis object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY implement terminal display for analysis object.
0005 %
0006 % CALL:      txt = display(ao)
0007 %          ao                                % without a semicolon
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
```

```

0010 %                >> ao.getInfo('display')
0011 %
0012 %                Get information about a specified set-plist by calling:
0013 %                >> ao.getInfo('display', 'None')
0014 %
0015 % VERSION:      $Id: display.m,v 1.30 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      31-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%
0021 %
0022 function varargout = display(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 import utils.const.*
0031 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039
0040 % Loop over AOs
0041 txt = {};
0042 for j=1: numel(as)
0043     banner_start = sprintf('----- ao: %s -----', ao_invars{j});
0044
0045     txt{end+1} = banner_start;
0046     txt{end+1} = ' ';
0047     txt{end+1} = sprintf('      name: %s', as(j).name);
0048     txt{end+1} = sprintf('      creator: %s', char(as(j).creator));
0049     txt{end+1} = sprintf('description: %s', char(as(j).description));
0050     if isempty(as(j).data)
0051         txt{end+1} = sprintf('      data: None');
0052     else
0053         mi = min(5, numel(as(j).data.y));
0054         dtxt = sprintf('      data:');
0055         if ~isa(as(j).data, 'cdata')
0056             for k=1:mi
0057                 dtxt = [dtxt sprintf(' (%s,%s)', mat2str(as(j).data.getX(k)),
mat2str(as(j).data.y(k)))];
0058             end
0059             if mi < length(as(j).data.y)
0060                 dtxt = [dtxt ' ...'];
0061             end
0062         else
0063             dtxt = [dtxt sprintf(' %s', mat2str(as(j).data.getY))];
0064             if length(dtxt) > 44
0065                 dtxt = [dtxt(1:44) '...'];
0066             end
0067         end
0068         txt{end+1} = dtxt;
0069
0070         % Add some data info
0071         w = display(as(j).data);
0072         for k=1: numel(w)
0073             txt{end+1} = sprintf('      %s', w{k});
0074         end
0075     end
0076     if isempty(as(j).hist)
0077         txt{end+1} = sprintf('      hist: None');
0078     else
0079         info = as(j).hist.methodInfo;
0080         txt{end+1} = sprintf('      hist: %s / %s / %s', info.mclass, info.mname,
info.mversion);
0081     end
0082     txt{end+1} = sprintf('      mfilename: %s', as(j).mfilename);
0083     txt{end+1} = sprintf('      mdlfilename: %s', as(j).mdlfilename);
0084
0085     banner_end(1:length(banner_start)) = '-';
0086     txt{end+1} = banner_end;
0087
0088     txt{end+1} = ' ';
0089     txt{end+1} = ' ';
0090 end
0091
0092 if nargout == 0
0093     for ii=1:length(txt)

```

```
0094         disp(txt{ii});
0095     end
0096 else
0097     varargout{1} = txt;
0098 end
0099 end
0100
0101 %-----
0102 % Get Info Object
0103 %-----
0104 function ii = getInfo(varargin)
0105     if nargin == 1 && strcmpi(varargin{1}, 'None')
0106         sets = {};
0107         pl = [];
0108     else
0109         sets = {'Default'};
0110         pl = getDefaultPlist;
0111     end
0112     % Build info object
0113     ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: display.m,v 1.30
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0114 end
0115
0116 %-----
0117 % Get Default Plist
0118 %-----
0119 function pl = getDefaultPlist()
0120     pl = plist();
0121 end
0122
0123
```

export

PURPOSE ^

EXPORT export an analysis object to a text file.

SYNOPSIS ^

function varargout = export(varargin)

DESCRIPTION ^

```
EXPORT export an analysis object to a text file.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: EXPORT export an analysis object to a text file.

CALL:      export(a, 'blah.txt');
           export(a, plist);

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('export')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('export', 'None')

VERSION:    $Id: export.m,v 1.15 2008/09/05 11:05:29 ingo Exp $

HISTORY:    30-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [imag](#) IMAG overloads the imaginary operator for Analysis objects.
-  [real](#) REAL overloads the real operator for Analysis objects.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % EXPORT export an analysis object to a text file.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: EXPORT export an analysis object to a text file.
0005 %
0006 % CALL:      export(a, 'blah.txt');
0007 %           export(a, plist);
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('export')
0011 %
0012 %              Get information about a specified set-plist by calling:
```



```

0013 %                >> ao.getInfo('export', 'None')
0014 %
0015 % VERSION:      $Id: export.m,v 1.15 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:      30-03-07 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = export(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 import utils.const.*
0031 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039 [pl, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0040
0041 % Combine plists
0042 pl = combine(pl, getDefaultPlist);
0043
0044 % get filename from plist
0045 filename = find(pl, 'filename');
0046 if isempty(filename)
0047     for jj = 1:numel(rest)
0048         if ischar(rest{jj})
0049             filename = rest{jj};
0050             break
0051         end
0052     end
0053 end
0054 if isempty(filename)
0055     error('### You must specify a filename. Either directly, or in a plist');
0056 end
0057
0058 if length(as) > 1
0059     error('### Export can only deal with one AO at a time.');
```


extractm

PURPOSE ^

EXTRACTM extracts an m-file from an analysis object and saves it to disk.

SYNOPSIS ^

function varargout = extractm(varargin)

DESCRIPTION ^

```
EXTRACTM extracts an m-file from an analysis object and saves it to disk.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: EXTRACTM extracts an m-file from an analysis object and saves it to
disk.

CALL:      extractm(a)

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('extractm')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('extractm', 'None')

VERSION:    $Id: extractm.m,v 1.7 2008/09/05 11:05:29 ingo Exp $

HISTORY:    23-02-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % EXTRACTM extracts an m-file from an analysis object and saves it to disk.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: EXTRACTM extracts an m-file from an analysis object and saves it to
0005 % disk.
0006 %
0007 % CALL:      extractm(a)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('extractm')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('extractm', 'None')
0014 %
0015 % VERSION:    $Id: extractm.m,v 1.7 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:    23-02-08 M Hewitson
0018 %              Creation
0019 %
```

```

0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = extractm(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 import utils.const.*
0031 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039
0040 % Loop over AOs
0041 for jj=1:numel(as)
0042     if isempty(as(j).mfile)
0043         warning('!!! No mfile found in %s. Skipping.', ao_invars{jj});
0044     end
0045     if isempty(as(j).mfilename)
0046         warning('!!! No filename found in %s. Skipping.', ao_invars{jj});
0047     end
0048     % split mdl contents by \n in to cell array
0049     mfile = regexp(as(j).mfile, '\\n', 'split');
0050     % reformat into a big string
0051     txt = sprintf([repmat('%s\t',1,size(mfile,1)),'\n'],mfile{:});
0052     txt = strrep(txt, '\t', '\n');
0053     % write string to disk
0054     fd = fopen(as(j).mfilename, 'w+');
0055     fwrite(fd, txt);
0056     fclose(fd);
0057
0058     utils.helper.msg(msg.PROC1, 'saved m-file to %s', as(j).mfilename);
0059 end
0060
0061 % Set output
0062 if nargout > 0
0063     varargout{1} = as;
0064 end
0065 end
0066
0067 %-----
0068 % Get Info Object
0069 %-----
0070 function ii = getInfo(varargin)
0071     if nargin == 1 && strcmpi(varargin{1}, 'None')
0072         sets = {};
0073         pl = [];
0074     else
0075         sets = {'Default'};
0076         pl = getDefaultPlist;
0077     end
0078     % Build info object
0079     ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: extractm.m,v 1.7
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0080 end
0081
0082 %-----
0083 % Get Default Plist
0084 %-----
0085 function pl_default = getDefaultPlist()
0086     pl_default = plist();
0087 end
0088

```

extractmdl

PURPOSE ^

EXTRACTMDL extracts an mdl file from an analysis object and saves it to disk.

SYNOPSIS ^

function varargout = extractmdl(varargin)

DESCRIPTION ^

```
EXTRACTMDL extracts an mdl file from an analysis object and saves it to disk.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: EXTRACTMDL extracts an mdl file from an analysis object and saves
             it to disk.

CALL:       extractmdl(a)

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('extractmdl')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('extractmdl', 'None')

VERSION:     $Id: extractmdl.m,v 1.7 2008/09/05 11:05:29 ingo Exp $

HISTORY:     31-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % EXTRACTMDL extracts an mdl file from an analysis object and saves it to disk.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: EXTRACTMDL extracts an mdl file from an analysis object and saves
0005 %              it to disk.
0006 %
0007 % CALL:       extractmdl(a)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> ao.getInfo('extractmdl')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> ao.getInfo('extractmdl', 'None')
0014 %
0015 % VERSION:     $Id: extractmdl.m,v 1.7 2008/09/05 11:05:29 ingo Exp $
0016 %
0017 % HISTORY:     31-03-07 M Hewitson
0018 %              Creation
0019 %
```

```

0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = extractmdl(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 import utils.const.*
0031 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039
0040 % Loop over AOs
0041 for jj=1:numel(as)
0042     if isempty(as(j).mdlfile)
0043         warning('!!! No model found in %s. Skipping.', ao_invars{jj});
0044     end
0045     if isempty(as(j).mdlfilename)
0046         warning('!!! No model filename found in %s. Skipping.', ao_invars{jj});
0047     end
0048     % split mdl contents by \n in to cell array
0049     mdlfile = regexp(as(j).mdlfile, '\\n', 'split');
0050     % reformat into a big string
0051     txt = sprintf([repmat('%s\\t',1,size(mdlfile,1)), '\\n'],mdlfile{:});
0052     txt = strrep(txt, '\\t', '\\');
0053     % write string to disk
0054     fd = fopen(as(j).mdlfilename, 'w+');
0055     fwrite(fd, txt);
0056     fclose(fd);
0057     utils.helper.msg(msg.PROC1, 'saved model to %s', as(j).mdlfilename);
0058 end
0059
0060 % Set output
0061 if nargout > 0
0062     varargout{1} = as;
0063 end
0064 end
0065
0066 %-----
0067 % Get Info Object
0068 %-----
0069 function ii = getInfo(varargin)
0070     if nargin == 1 && strcmpi(varargin{1}, 'None')
0071         sets = {};
0072         pl = [];
0073     else
0074         sets = {'Default'};
0075         pl = getDefaultPlist;
0076     end
0077     % Build info object
0078     ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: extractmdl.m,v
1.7 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0079 end
0080
0081 %-----
0082 % Get Default Plist
0083 %-----
0084 function pl_default = getDefaultPlist()
0085     pl_default = plist();
0086 end
0087
0088

```

iplot

PURPOSE ^

IPLLOT provides an intelligent plotting tool for LTPDA.

SYNOPSIS ^

function varargout = iplot(varargin)

DESCRIPTION ^

IPLLOT provides an intelligent plotting tool for LTPDA.
%%%

DESCRIPTION: IPLLOT provides an intelligent plotting tool for LTPDA.

CALL:
 hfig = iplot (a,pl)
 [hfig, hax] = iplot (a,pl)
 [hfig, hax, hli] = iplot (a,pl)

INPUTS: pl - a parameter list
 a - input analysis object

OUTPUTS: hfig - handles to figures
 hax - handles to axes
 hli - handles to lines

Plot parameters:

 'Arrangement' - select the plot layout:
 'single' - plots all AOs on individual figures
 'stacked' - plots all AOs on the same axes [default]
 'subplots' - plots all AOs on subplots

 'Function' - specify the plot function:
 'plot', 'stairs', 'stem'
 [default: 'plot']
 *** doesn't work for xyzdata AOs

Line parameters:

The following properties take cell array values. If the length of the cell array is shorter than the number of lines to plot, the remaining lines will be plotted with the default options. If the cell array is of length 2 and the first cell contains the string 'all', then the second cell is used to set the property of all lines.

 'LineColors' - a cell array of color definitions for each line.

 'LineStyle' - a cell array of line styles.

 'Markers' - a cell array of markers.

 'LineWidths' - a cell array of line widths. If the length of the cell array is shorter than the number of lines to plot, the remaining lines will be plotted with the default line width.

Axes parameters:

 'Legends' - specify a cell array of strings to be used for the plot legends. If a cell contains an empty string, the default legend string is built. If a single string 'off' is given instead of a cell array, then the legends are all switched off.

 'XLabels' - Specify the labels to be used on the x-axis. The units are added from the data object 'xunits' property.

 'YLabels' - Specify the labels to be used on the y-axis. The

```
units are added from the data object 'yunits'
property. If the object contains complex data,
you should specify two y-labels for that object.

The following axis properties also work with the 'all' keyword as
described above in the line properties section.

'XScales' - Specify the scales to be used on the x-axes.
'YScales' - Specify the scales to the used on the y-axes. If
an object contains complex data, you should
specify two y-labels for that object.
'XRanges' - Specify the ranges to be displayed on the x-axes.
'YRanges' - Specify the ranges to the displayed on the
y-axes.

EXAMPLES:

1) Plot two time-series AOs with different colors, line styles, and widths
pl = plist('Linecolors', {'g', 'k'}, 'LineStyles', {'', '--'}, 'LineWidths', {1, 4});
iplot(tsao1, tsao2, pl);

2) Plot two time-series AOs in subplots. Also override the second legend
text and the first line style.
pl = plist('Arrangement', 'subplots', 'LineStyles', {'--'}, 'Legends', {'', 'My Sine
Wave'});
iplot(tsao1, tsao2, pl);

3) Plot two time-series AOs taking the square of the y-values of the
first AO and the log of the x-values of the second AO.
pl = plist('Arrangement', 'subplots', 'YMaths', 'y.^2', 'XMaths', {'', 'log(x)'});
iplot(tsao1, tsao2, pl);

4) Plot two frequency-series AOs on subplots with the same Y-scales and
Y-ranges
pl1 = plist('Yscales', {'All', 'lin'});
pl2 = plist('arrangement', 'subplots', 'YRanges', {'All', [1e-6 100]});
iplot(fsd1, fsd2, pl1, pl2)

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('iplot')

Get information about a specified set-plist by calling:
>> ao.getInfo('iplot', 'None')

VERSION: $Id: iplot.m,v 1.48 2008/09/05 14:14:33 hewitson Exp $

HISTORY: 22-12-07 M Hewitson
Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [char](#) CHAR overloads char() function for analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [imag](#) IMAG overloads the imaginary operator for Analysis objects.
-  [real](#) REAL overloads the real operator for Analysis objects.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

 function varargout = fs_plot(varargin)
 function varargout = xy_plot(varargin)
 function varargout = y_plot(varargin)
 function varargout = xyz_plot(varargin)
 function ii = getInfo(varargin)
 function out = getDefaultPlist(set)
 function ss = fixlabel(ss)

SOURCE CODE ^

```

0001 % IPLOT provides an intelligent plotting tool for LTPDA.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: IPLOT provides an intelligent plotting tool for LTPDA.
0005 %
0006 % CALL:          hfig = iplot (a,pl)
0007 %               [hfig, hax] = iplot (a,pl)
0008 %               [hfig, hax, hli] = iplot (a,pl)
0009 %
0010 % INPUTS:        pl   - a parameter list
0011 %                a    - input analysis object
0012 %
0013 % OUTPUTS:       hfig - handles to figures
0014 %                hax  - handles to axes
0015 %                hli  - handles to lines
0016 %
0017 % Plot parameters:
0018 %
0019 %               'Arrangement' - select the plot layout:
0020 %                               'single' - plots all AOs on individual figures
0021 %                               'stacked' - plots all AOs on the same axes [default]
0022 %                               'subplots' - plots all AOs on subplots
0023 %
0024 %               'Function'    - specify the plot function:
0025 %                               'plot' 'stairs' 'stem'
0026 %                               [default: 'plot']
0027 %                               *** doesn't work for xyzdata AOs
0028 %
0029 % Line parameters:
0030 %
0031 %               The following properties take cell array values. If the length of
0032 %               the cell array is shorter than the number of lines to plot, the
0033 %               remaining lines will be plotted with the default options. If the
0034 %               cell array is of length 2 and the first cell contains the string
0035 %               'all', then the second cell is used to set the property of all
0036 %               lines.
0037 %
0038 %               'LineColors' - a cell array of color definitions for each line.
0039 %
0040 %               'LineStyles' - a cell array of line styles.
0041 %
0042 %               'Markers'    - a cell array of markers.
0043 %
0044 %               'LineWidths' - a cell array of line widths. If the length of the
0045 %               cell array is shorter than the number of lines to
0046 %               plot, the remaining lines will be plotted with
0047 %               the default line width.
0048 %
0049 % Axes parameters:
0050 %
0051 %               'Legends'    - specify a cell array of strings to be used for
0052 %               the plot legends. If a cell contains an empty
0053 %               string, the default legend string is built.
0054 %               If a single string 'off' is given instead of a
0055 %               cell array, then the legends are all switched
0056 %               off.
0057 %
0058 %               'XLabels'    - Specify the labels to be used on the x-axis. The
0059 %               units are added from the data object 'xunits'
0060 %               property.
0061 %
0062 %               'YLabels'    - Specify the labels to be used on the y-axis. The
0063 %               units are added from the data object 'yunits'
0064 %               property. If the object contains complex data,
0065 %               you should specify two y-labels for that object.
0066 %
0067 %               The following axis properties also work with the 'all' keyword as
0068 %               described above in the line properties section.
0069 %
  
```

```

0070 %           'Xscales' - Specify the scales to be used on the x-axes.
0071 %
0072 %           'Yscales' - Specify the scales to be used on the y-axes. If
0073 %                   an object contains complex data, you should
0074 %                   specify two y-labels for that object.
0075 %
0076 %           'XRanges' - Specify the ranges to be displayed on the x-axes.
0077 %
0078 %           'YRanges' - Specify the ranges to be displayed on the
0079 %                   y-axes.
0080 %
0081 % EXAMPLES:
0082 %
0083 % 1) Plot two time-series AOs with different colors, line styles, and widths
0084 %
0085 %     pl = plist('Linecolors', {'g', 'k'}, 'LineStyles', {'', '--'}, 'LineWidths', {1,
0086 % 4});
0087 %     iplot(tsao1, tsao2, pl);
0088 %
0089 % 2) Plot two time-series AOs in subplots. Also override the second legend
0090 %     text and the first line style.
0091 %
0092 %     pl = plist('Arrangement', 'subplots', 'LineStyles', {'--'}, 'Legends', {'', 'My Sine
0093 % Wave'});
0094 %     iplot(tsao1, tsao2, pl);
0095 %
0096 % 3) Plot two time-series AOs taking the square of the y-values of the
0097 %     first AO and the log of the x-values of the second AO.
0098 %
0099 %     pl = plist('Arrangement', 'subplots', 'YMaths', 'y.^2', 'XMaths', {'', 'log(x)'});
0100 %     iplot(tsao1, tsao2, pl);
0101 %
0102 % 4) Plot two frequency-series AOs on subplots with the same Y-scales and
0103 %     Y-ranges
0104 %
0105 %     pl1 = plist('Yscales', {'All', 'lin'});
0106 %     pl2 = plist('arrangement', 'subplots', 'YRanges', {'All', [1e-6 100]});
0107 %     iplot(fsd1, fsd2, pl1, pl2)
0108 %
0109 % M-FILE INFO: Get information about this methods by calling
0110 %             >> ao.getInfo('iplot')
0111 %
0112 %             Get information about a specified set-plist by calling:
0113 %             >> ao.getInfo('iplot', 'None')
0114 %
0115 % VERSION:     $Id: iplot.m,v 1.48 2008/09/05 14:14:33 hewitson Exp $
0116 %
0117 % HISTORY:     22-12-07 M Hewitson
0118 %             Creation
0119 %%%%%%%%%%%
0120 %%%%%%%%%%%
0121 %%%%%%%%%%%
0122 %%%%%%%%%%%
0123 %
0124 % TODO:
0125 %     1) Add XRange, YRange, ZRange to xyzdata
0126 %     5) How do we deal with axis units? Symbolic math toolbox?
0127 %     7) Should we have a plot_plist field in AO? Then this can be combined
0128 %         with the input plist.
0129 %
0130 %
0131 function varargout = iplot(varargin)
0132
0133 import utils.const.*
0134
0135 %% Check if this is a call for parameters
0136 if utils.helper.isinfocall(varargin{:})
0137     varargout{1} = getInfo(varargin{3});
0138     return
0139 end
0140
0141 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0142
0143 % Collect input variable names
0144 in_names = cell(size(varargin));
0145 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0146
0147 % Collect all AOs and plists
0148 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0149 [upl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0150
0151 %% Go through AOs and collect them into similar types
0152
0153 tsAOs = [];

```

```

0154 fsAOs = [];
0155 xyAOs = [];
0156 xyzAOs = [];
0157 cAOs = [];
0158
0159 for j=1:numel(as)
0160     switch class(as(j).data)
0161         case 'tsdata'
0162             tsAOs = [tsAOs as(j)];
0163         case 'fsdata'
0164             fsAOs = [fsAOs as(j)];
0165         case 'xydata'
0166             xyAOs = [xyAOs as(j)];
0167         case 'xyzdata'
0168             xyzAOs = [xyzAOs as(j)];
0169         case 'cdata'
0170             cAOs = [cAOs as(j)];
0171         otherwise
0172             warning('!!! Unknown data type %s', class(as(j).data));
0173     end
0174 end
0175
0176 %% Now plot all the objects on separate figures
0177
0178 hfig = [];
0179 hax = [];
0180 hli = [];
0181
0182 %----- TSDATA
0183 if ~isempty(tsAOs)
0184     % get default plist
0185     dpl = getDefaultPlist('Time-series plot');
0186     % combine the plists
0187     pl = combine(upl, dpl);
0188     % Call x-y plot
0189     [hf, ha, hl] = xy_plot(tsAOs, pl);
0190     hfig = [hfig hf];
0191     hax = [hax ha];
0192     hli = [hli hl];
0193 end
0194 %----- XYDATA
0195 if ~isempty(xyAOs)
0196     % get default plist
0197     dpl = getDefaultPlist('X-Y data plot');
0198     % combine the plists
0199     pl = combine(upl, dpl);
0200     % Call x-y plot
0201     [hf, ha, hl] = xy_plot(xyAOs, pl);
0202     hfig = [hfig hf];
0203     hax = [hax ha];
0204     hli = [hli hl];
0205 end
0206 %----- XYZDATA
0207 if ~isempty(xyzAOs)
0208     % get default plist
0209     dpl = getDefaultPlist('3D plot');
0210     % combine the plists
0211     pl = combine(upl, dpl);
0212     % Call x-y-z plot
0213     [hf, ha, hl] = xyz_plot(xyzAOs, pl);
0214     hfig = [hfig hf];
0215     hax = [hax ha];
0216     hli = [hli hl];
0217 end
0218 %----- CDATA
0219 if ~isempty(cAOs)
0220     % get default plist
0221     dpl = getDefaultPlist('Y data plot');
0222     % combine the plists
0223     pl = combine(upl, dpl);
0224     % Call x-y plot
0225     [hf, ha, hl] = y_plot(cAOs, pl);
0226     hfig = [hfig hf];
0227     hax = [hax ha];
0228     hli = [hli hl];
0229 end
0230 %----- FSDATA
0231 if ~isempty(fsAOs)
0232     % get default plist
0233     dpl = getDefaultPlist('Frequency-series plot');
0234     % combine the plists
0235     pl = combine(upl, dpl);
0236     % Call fsdata plot
0237     [hf, ha, hl] = fs_plot(fsAOs, pl);
0238     hfig = [hfig hf];
0239     hax = [hax ha];

```

```

0240     hli = [hli hl];
0241 end
0242
0243 %% Deal with outputs
0244 if nargout == 1
0245     varargout{1} = hfig;
0246 end
0247 if nargout == 2
0248     varargout{1} = hfig;
0249     varargout{2} = hax;
0250 end
0251 if nargout == 3
0252     varargout{1} = hfig;
0253     varargout{2} = hax;
0254     varargout{3} = hli;
0255 end
0256
0257 if nargout > 3
0258     error('### Incorrect number of outputs');
0259 end
0260
0261 end
0262
0263 %-----
0264 % Plot fsdata objects
0265 %
0266 function varargout = fs_plot(varargin)
0267
0268     aos = varargin{1};
0269     pl = varargin{2};
0270
0271 % Extract parameters
0272 arrangement = find(pl, 'Arrangement');
0273 colors = find(pl, 'Colors');
0274 linecolors = find(pl, 'LineColors');
0275 linestyle = find(pl, 'LineStyle');
0276 markers = find(pl, 'Markers');
0277 linewidths = find(pl, 'LineWidths');
0278 legends = find(pl, 'Legends');
0279 ylabel = find(pl, 'YLabels');
0280 xlabel = find(pl, 'XLabels');
0281 yscale = find(pl, 'YScales');
0282 xscale = find(pl, 'XScale');
0283 yrange = find(pl, 'YRanges');
0284 xrange = find(pl, 'XRanges');
0285 xmaths = find(pl, 'XMaths');
0286 ymaths = find(pl, 'YMaths');
0287 type = find(pl, 'Function');
0288 legendLoc = find(pl, 'LegendLocation');
0289 complexPlotType = find(pl, 'complexPlotType');
0290
0291 % check whether we want legends or not
0292 if iscell(legends)
0293     legendsOn = 1;
0294 else
0295     if strcmpi(legends, 'off')
0296         legendsOn = 0;
0297     else
0298         legendsOn = 1;
0299         legends = [];
0300     end
0301 end
0302
0303 if ~iscell(linewidths), linewidths = {linewidths}; end
0304 if ~iscell(linestyle), linestyle = {linestyle}; end
0305 if ~iscell(linecolors), linecolors = {linecolors}; end
0306 if ~iscell(markers), markers = {markers}; end
0307 if ~iscell(legends), legends = {legends}; end
0308 if ~iscell(ylabel), ylabel = {ylabel}; end
0309 if ~iscell(xlabel), xlabel = {xlabel}; end
0310 if ~iscell(xmaths), xmaths = {xmaths}; end
0311 if ~iscell(ymaths), ymaths = {ymaths}; end
0312 if ~iscell(xscale), xscale = {xscale}; end
0313 if ~iscell(yscale), yscale = {yscale}; end
0314 if ~iscell(xrange), xrange = {xrange}; end
0315 if ~iscell(yrange), yrange = {yrange}; end
0316
0317 % collect figure handles
0318 tsfig = []; tsax = []; tsli = [];
0319 % Legend holder
0320 legendStr = [];
0321
0322 if ~isempty(aos)
0323     % Now loop over AOs
0324     Na = length(aos);
0325     % - first to check if any are complex

```

```

0326 haveComplex = 0;
0327 for j=1:Na
0328     y = aos(j).data.y;
0329     ymath = '';
0330     if j<=length(ymaths)
0331         if ~isempty(ymaths{j})
0332             eval(sprintf('y = %s;', ymaths{j}));
0333             ymath = ymaths{j};
0334         end
0335     end
0336     xmath = '';
0337     if j<=length(xmaths)
0338         if ~isempty(xmaths{j})
0339             eval(sprintf('x = %s;', xmaths{j}));
0340             xmath = xmaths{j};
0341         end
0342     end
0343     if ~isreal(y)
0344         haveComplex = 1;
0345     end
0346 end
0347
0348 % Loop over the AOs now
0349 for j=1:Na
0350     % set real and imag subplot handles to empty
0351     tsax_r = [];
0352     tsax_i = [];
0353     %----- Apply math functions
0354     x = aos(j).data.x;
0355     y = aos(j).data.y;
0356     ymath = '';
0357     if j<=length(ymaths)
0358         if ~isempty(ymaths{j})
0359             eval(sprintf('y = %s;', ymaths{j}));
0360             ymath = ymaths{j};
0361         end
0362     end
0363     xmath = '';
0364     if j<=length(xmaths)
0365         if ~isempty(xmaths{j})
0366             eval(sprintf('x = %s;', xmaths{j}));
0367             xmath = xmaths{j};
0368         end
0369     end
0370     % what figures do we need?
0371     switch arrangement
0372     case 'single'
0373         tsfig = [tsfig figure];
0374         col = colors{1};
0375         % check if this data set is real or complex
0376         if ~isreal(y)
0377             % complex means we use two subplots
0378             tsax_r = subplot(2,1,1);
0379             tsax_i = subplot(2,1,2);
0380             tsax = [tsax tsax_r tsax_i];
0381         else
0382             % real means we use a single subplot
0383             tsax_r = subplot(1, 1, 1);
0384             tsax = [tsax tsax_r];
0385         end
0386     case 'stacked'
0387         if j==1, tsfig = figure; end
0388         % if at least one of the input fsdata AOs is complex, we need to
0389         % allow for subplots
0390         if haveComplex
0391             tsax_r = subplot(2,1,1);
0392             tsax_i = subplot(2,1,2);
0393             tsax = [tsax_r tsax_i];
0394         else
0395             tsax_r = subplot(1, 1, 1);
0396             tsax = tsax_r;
0397         end
0398         col = colors{mod(j-1,length(colors))+1};
0399         hold(tsax_r, 'on');
0400         if ishandle(tsax_i)
0401             hold(tsax_i, 'on');
0402         end
0403     case 'subplots'
0404         if j == 1, tsfig = figure; end
0405         c = 1+(j-1)*2;
0406         sx = Na;
0407         sy = 2;
0408         % Now we have one or two subplots per input object.
0409         if ~isreal(y)
0410             tsax_r = subplot(sx, sy,c);
0411             tsax_i = subplot(sx, sy,c+1);

```

```

0412     tsax  = [tsax tsax_r tsax_i];
0413     else
0414         tsax_r = subplot(sx, sy, c:c+1);
0415         tsax  = [tsax tsax_r];
0416     end
0417     col = colors{1};
0418     otherwise
0419         error('### Unknown plot arrangement');
0420 end
0421
0422 %----- Plot the data
0423
0424 % plot real or complex data and setup default values for scale and
0425 % labels as we go.
0426 if isreal(y)
0427     li = feval(type, tsax_r, x, y);
0428     tsli = [tsli li];
0429     ylabelr = 'amplitude';
0430     ylabeli = 'imag';
0431     yscaleR = 'log';
0432     yscaleI = 'lin';
0433     xscaleR = 'log';
0434     xscaleI = 'lin';
0435 else
0436     switch complexPlotType
0437     case 'realimag'
0438         li = feval(type, tsax_r, x, y);
0439         li = [li feval(type, tsax_i, x, imag(y))];
0440         tsli = [tsli li];
0441         ylabelr = 'real';
0442         ylabeli = 'imag';
0443         yscaleR = 'lin';
0444         yscaleI = 'lin';
0445         xscaleR = 'lin';
0446         xscaleI = 'lin';
0447     case 'absdeg'
0448         li = feval(type, tsax_r, x, abs(y));
0449         li = [li feval(type, tsax_i, x, utils.math.phase(y))];
0450         tsli = [tsli li];
0451         ylabelr = 'amplitude';
0452         ylabeli = 'Phase [deg]';
0453         yscaleR = 'log';
0454         yscaleI = 'lin';
0455         xscaleR = 'log';
0456         xscaleI = 'log';
0457     case 'absrad'
0458         li = feval(type, tsax_r, x, abs(y));
0459         li = [li feval(type, tsax_i, x, angle(y))];
0460         tsli = [tsli li];
0461         ylabelr = 'amplitude';
0462         ylabeli = 'phase [rad]';
0463         yscaleR = 'log';
0464         yscaleI = 'lin';
0465         xscaleR = 'log';
0466         xscaleI = 'log';
0467     otherwise
0468         error('### Unknown plot type for complex data');
0469     end
0470 end
0471
0472 %----- Axis properties
0473
0474 % Set ylabel
0475 c = 1+(j-1)*2;
0476 if length(ylabels)==2 && strcmpi(ylabels{1}, 'all')
0477     ylstrR = ylabels{2};
0478 elseif c<=length(ylabels)
0479     if ~isempty(ylabels{c})
0480         ylstrR = ylabels{c};
0481     else
0482         ylstrR = ylabelr;
0483     end
0484 else
0485     ylstrR = ylabelr;
0486 end
0487 if c<length(ylabels)
0488     if ~isempty(ylabels{c+1})
0489         ylstrI = ylabels{c+1};
0490     else
0491         ylstrI = ylabeli;
0492     end
0493 else
0494     ylstrI = ylabeli;
0495 end
0496 if ~isempty(ymath)
0497     ymath = strrep(ymath, 'y', sprintf('%s', char(aos(j)).data.yunits));

```



```

0498     ylstrR = [ylstrR ' ' ymath ];
0499 else
0500     ylstrR = [ylstrR ' ' char(aos(j).data.yunits) ];
0501 end
0502 ylstrR = fixlabel(ylstrR);
0503 ylabel(tsax_r, ylstrR);
0504 if ishandle(tsax_i)
0505     ylstrI = fixlabel(ylstrI);
0506     ylabel(tsax_i, ylstrI);
0507 end
0508
0509 % Set xlabel
0510 if length(xlabels)==2 && strcmpi(xlabels{1}, 'all')
0511     xlstr = xlabels{2};
0512 elseif j<=length(xlabels) && ~isempty(xlabels{j})
0513     xlstr = xlabels{j};
0514 else
0515     xlstr = find(pl, 'XLabel');
0516 end
0517 if ~isempty(xmath)
0518     xmath = strrep(xmath, 'x', sprintf('%s', char(aos(j).data.xunits)));
0519     xlstr = [xlstr ' ' xmath ];
0520 else
0521     xlstr = [xlstr ' ' char(aos(j).data.xunits) ];
0522 end
0523 xlstr = fixlabel(xlstr);
0524 xlabel(tsax_r, xlstr);
0525 if ishandle(tsax_i)
0526     xlabel(tsax_i, xlstr);
0527 end
0528
0529 % Set grid on or off
0530 grid(tsax_r, 'on');
0531 if ishandle(tsax_i)
0532     grid(tsax_i, 'on');
0533 end
0534
0535 % Set Y scale
0536 if length(yscales)==2 && strcmpi(yscales{1}, 'all')
0537     yscaleR = yscales{2};
0538     yscaleI = yscales{2};
0539 else
0540     c = 1+(j-1)*2;
0541     if c<=length(yscales)
0542         if ~isempty(yscales{c})
0543             yscaleR = yscales{c};
0544         end
0545     end
0546     if c<length(yscales)
0547         if ~isempty(yscales{c+1})
0548             yscaleI = yscales{c+1};
0549         end
0550     end
0551 end
0552
0553 set(tsax_r, 'YScale', yscaleR);
0554 if ishandle(tsax_i)
0555     set(tsax_i, 'YScale', yscaleI);
0556 end
0557
0558 % Set X scale
0559 if length(xscales)==2 && strcmpi(xscales{1}, 'all')
0560     xscaleR = xscales{2};
0561     xscaleI = xscales{2};
0562 else
0563     c = 1+(j-1)*2;
0564     if c<=length(xscales)
0565         if ~isempty(xscales{c})
0566             xscaleR = xscales{c};
0567         end
0568     end
0569     if c<length(xscales)
0570         if ~isempty(xscales{c+1})
0571             xscaleI = xscales{c+1};
0572         end
0573     end
0574 end
0575
0576 set(tsax_r, 'XScale', xscaleR);
0577 if ishandle(tsax_i)
0578     set(tsax_i, 'XScale', xscaleI);
0579 end
0580
0581 % Set Y range
0582 if length(yranges)==2 && strcmpi(yranges{1}, 'all')
0583     set(tsax_r, 'YLim', yranges{2});
0584     if ishandle(tsax_i)

```

```

0584         set(tsax_i, 'YLim', yranges{2});
0585     end
0586 else
0587     c = 1+(j-1)*2;
0588     if c<=length(yranges)
0589         if ~isempty(yranges{c})
0590             set(tsax_r, 'YLim', yranges{c});
0591         end
0592     end
0593     if c<length(yranges)
0594         if ~isempty(yranges{c+1})
0595             if ishandle(tsax_i)
0596                 set(tsax_i, 'YLim', yranges{c+1});
0597             end
0598         end
0599     end
0600 end
0601
0602 % Set X range
0603 if length(xranges)==2 && strcmpi(xranges{1}, 'all')
0604     set(tsax_r, 'XLim', xrange{2});
0605     if ishandle(tsax_i)
0606         set(tsax_i, 'XLim', xrange{2});
0607     end
0608 else
0609     c = 1+(j-1)*2;
0610     if c<=length(xranges)
0611         if ~isempty(xranges{c})
0612             set(tsax_r, 'XLim', xrange{c});
0613         end
0614     end
0615     if c<length(xranges)
0616         if ~isempty(xranges{c+1})
0617             if ishandle(tsax_i)
0618                 set(tsax_i, 'XLim', xrange{c+1});
0619             end
0620         end
0621     end
0622 end
0623
0624 %----- line properties
0625
0626 % Set line color
0627 if isreal(y)
0628     set(li, 'Color', col);
0629 else
0630     set(li, 'Color', col);
0631     set(li, 'Color', col);
0632 end
0633
0634 % Override with user colors
0635 if length(linecolors) == 2 && strcmpi(linecolors{1}, 'all')
0636     set(li, 'Color', linecolors{2});
0637 else
0638     if j<=length(linecolors) && ~isempty(linecolors{j})
0639         if isreal(y)
0640             set(li, 'Color', linecolors{j});
0641         else
0642             set(li, 'Color', linecolors{j});
0643             set(li, 'Color', linecolors{j});
0644         end
0645     end
0646 end
0647
0648 % Set line style
0649 if length(linestyles) == 2 && strcmpi(linestyles{1}, 'all')
0650     set(li, 'LineStyle', linestyles{2});
0651 else
0652     if j<=length(linestyles) && ~isempty(linestyles{j})
0653         if isreal(y)
0654             set(li, 'LineStyle', linestyles{j});
0655         else
0656             set(li, 'LineStyle', linestyles{j});
0657             set(li, 'LineStyle', linestyles{j});
0658         end
0659     end
0660 end
0661
0662 % Set line widths
0663 if length(linewidths) == 2 && strcmpi(linewidths{1}, 'all')
0664     set(li, 'LineWidth', linewidths{2});
0665 else
0666     if j<=length(linewidths) && ~isempty(linewidths{j})
0667         if isreal(y)
0668             set(li, 'LineWidth', linewidths{j});
0669         else

```



```

0670         set(li, 'LineWidth', linewidths{j});
0671         set(li, 'LineWidth', linewidths{j});
0672     end
0673 end
0674 end
0675
0676 % Set markers
0677 if length(markers) == 2 && strcmpi(markers{1}, 'all')
0678     set(li, 'Marker', markers{2});
0679 else
0680     if j<=length(markers) && ~isempty(markers{j})
0681         if isreal(y)
0682             set(li, 'Marker', markers{j});
0683         else
0684             set(li, 'Marker', markers{j});
0685             set(li, 'Marker', markers{j});
0686         end
0687     end
0688 end
0689
0690 % Set legend string
0691 lstr = '';
0692 if legendsOn
0693     if j<=length(legends) && ~isempty(legends{j})
0694         lstr = legends{j};
0695     else
0696         lstr = utils.plottools.label(aos(j).name);
0697     end
0698 end
0699 legendStr = [legendStr cellstr(lstr)];
0700
0701 % Set the legend now if we can
0702 if legendsOn
0703     if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
0704         legend(tsax_r, fixlabel(legendStr{end}), 'Location', legendLoc);
0705     end
0706 end
0707 end % End loop over AOs
0708
0709 % Process legends for stacked plots
0710 if legendsOn
0711     if strcmp(arrangement, 'stacked')
0712         h = legend(tsax_r, fixlabel(legendStr), 'Location', legendLoc);
0713         set(h, 'FontSize', 10)
0714     end
0715 end
0716 end % End empty AOs
0717
0718 % Set outputs
0719 if nargout > 0
0720     varargout{1} = tsfig;
0721 end
0722 if nargout > 1
0723     varargout{2} = tsax;
0724 end
0725 if nargout == 3
0726     varargout{3} = tsli;
0727 end
0728 if nargout > 3
0729     error('### Too many output arguments');
0730 end
0731 end % End fs_plot
0732
0733 %-----
0734 % Plot tsdata and xydata objects
0735 %
0736 function varargout = xy_plot(varargin)
0737
0738     aos = varargin{1};
0739     pl = varargin{2};
0740
0741 % Extract parameters
0742 arrangement = find(pl, 'Arrangement');
0743 linecolors = find(pl, 'LineColors');
0744 colors = find(pl, 'Colors');
0745 linestyle = find(pl, 'LineStyles');
0746 markers = find(pl, 'Markers');
0747 linewidths = find(pl, 'Line1Widths');
0748 legends = find(pl, 'Legends');
0749 ylabels = find(pl, 'YLabels');
0750 xlabel = find(pl, 'XLabels');
0751 xmaths = find(pl, 'XMaths');
0752 ymaths = find(pl, 'YMaths');
0753 yranges = find(pl, 'YRanges');
0754 xrange = find(pl, 'XRanges');
0755 yscale = find(pl, 'YScales');

```

```

0756 xscales      = find(pl, 'Xscales');
0757 type        = find(pl, 'Function');
0758 legendLoc   = find(pl, 'LegendLocation');
0759
0760 % check whether we want legends or not
0761 if iscell(legends)
0762     legendsOn = 1;
0763 else
0764     if strcmpi(legends, 'off')
0765         legendsOn = 0;
0766     else
0767         legendsOn = 1;
0768         legends = [];
0769     end
0770 end
0771
0772 if ~iscell(linewidths), linewidths = {linewidths}; end
0773 if ~iscell(linestyles), linestyles = {linestyles}; end
0774 if ~iscell(linecolors), linecolors = {linecolors}; end
0775 if ~iscell(markers), markers = {markers}; end
0776 if ~iscell(legends), legends = {legends}; end
0777 if ~iscell(ylabels), ylabels = {ylabels}; end
0778 if ~iscell(xlabels), xlabels = {xlabels}; end
0779 if ~iscell(xmaths), xmaths = {xmaths}; end
0780 if ~iscell(ymaths), ymaths = {ymaths}; end
0781 if ~iscell(xranges), xrange = {xranges}; end
0782 if ~iscell(yranges), yrange = {yranges}; end
0783 if ~iscell(xscales), xscales = {xscales}; end
0784 if ~iscell(yscales), yscales = {yscales}; end
0785
0786 % collect figure handles
0787 tsfig = []; tsax = []; tsli = [];
0788 % Legend holder
0789 legendStr = [];
0790 if ~isempty(aos)
0791     % Now loop over AOs to get earliest start time
0792     if strcmp(arrangement, 'stacked')
0793         T0 = 1e50;
0794         Na = length(aos);
0795         for jj=1:Na
0796             % Get this AO
0797             if isa(aos(jj).data, 'tsdata')
0798                 if aos(jj).data.t0.utc_epoch_milli/1000 < T0
0799                     T0 = floor(aos(jj).data.t0.utc_epoch_milli/1000);
0800                 end
0801             end
0802         end
0803     else
0804         T0 = 0;
0805     end
0806
0807 % Now loop over AOs
0808 Na = length(aos);
0809 for j=1:Na
0810     % Get this AO
0811     toff = 0;
0812     % what figures do we need?
0813     switch arrangement
0814     case 'single'
0815         tsfig = [tsfig figure];
0816         tsax = subplot(1,1,1);
0817         col = colors{1};
0818         if isa(aos(j).data, 'tsdata')
0819             torigin = aos(j).data.t0;
0820         end
0821     case 'stacked'
0822         if j==1, tsfig = figure; end
0823         tsax = subplot(1,1,1);
0824         col = colors{mod(j-1, length(colors))+1};
0825         hold on;
0826         % deal with time-stamps here
0827         if isa(aos(j).data, 'tsdata')
0828             toff = aos(j).data.t0.utc_epoch_milli/1000 - T0;
0829         else
0830             toff = 0;
0831         end
0832         if isa(aos(j).data, 'tsdata')
0833             torigin = time(T0*1000);
0834         end
0835     case 'subplots'
0836         if j == 1, tsfig = figure; end
0837         tsax = [tsax subplot(Na, 1, j)];
0838         col = colors{1};
0839         if isa(aos(j).data, 'tsdata')
0840             torigin = aos(j).data.t0;
0841         end
0842     end

```

```

0842         otherwise
0843             error('### Unknown plot arrangement');
0844     end
0845
0846     %----- Apply math functions
0847
0848     % need to offset for this time-series
0849     x = aos(j).data.getX + toff;
0850     y = aos(j).data.y;
0851
0852     ymath = '';
0853     if j<=length(ymaths)
0854         if ~isempty(ymaths{j})
0855             eval(sprintf('y = %s;', ymaths{j}));
0856             ymath = ymaths{j};
0857         end
0858     end
0859     xmath = '';
0860     if j<=length(xmaths)
0861         if ~isempty(xmaths{j})
0862             eval(sprintf('x = %s;', xmaths{j}));
0863             xmath = xmaths{j};
0864         end
0865     end
0866
0867     %----- Plot the data
0868
0869     li = feval(type, tsax(end), x, y);
0870     tsli = [tsli li];
0871     if isa(aos(j).data, 'tsdata')
0872         title(sprintf('Time origin: %s', char(torigin)));
0873     end
0874
0875     %----- Axis properties
0876
0877     % Set ylabel
0878     if j<=length(ylabels) && ~isempty(ylabels{j})
0879         ylstr = ylabels{j};
0880     else
0881         ylstr = find(pl, 'YLabel');
0882     end
0883     if ~isempty(ymath)
0884         ymath = strrep(ymath, 'y', sprintf('%s', char(aos(j).data.yunits)));
0885         ylstr = [ylstr ' ' ymath];
0886     else
0887         ylstr = [ylstr ' ' char(aos(j).data.yunits)];
0888     end
0889     ylstr = fixlabel(ylstr);
0890     ylabel(ylstr);
0891
0892     % Set xlabel
0893     if j<=length(xlabels) && ~isempty(xlabels{j})
0894         xlstr = xlabels{j};
0895     else
0896         xlstr = find(pl, 'XLabel');
0897     end
0898     if ~isempty(xmath)
0899         xmath = strrep(xmath, 'x', sprintf('%s', char(aos(j).data.xunits)));
0900         xlstr = [xlstr ' ' xmath];
0901     else
0902         xlstr = [xlstr ' ' char(aos(j).data.xunits)];
0903     end
0904     xlstr = fixlabel(xlstr);
0905     xlabel(xlstr);
0906
0907     % Set Y range
0908     if length(yranges) == 2 && strcmpi(yranges{1}, 'all')
0909         set(tsax(end), 'YLim', yranges{2});
0910     else
0911         if j<=length(yranges)
0912             if ~isempty(yranges{j})
0913                 set(tsax(end), 'YLim', yranges{j});
0914             end
0915         end
0916     end
0917
0918     % Set Y scale
0919     yscaler = 'lin';
0920     if length(yscales) == 2 && strcmpi(yscales{1}, 'all')
0921         yscaler = yscales{2};
0922     else
0923         if j<=length(yscales)
0924             if ~isempty(yscales{j})
0925                 yscaler = yscales{j};
0926             end
0927         end
0928     end

```

```

0928     set(tsax(end), 'YScale', yscaleR);
0929
0930     % Set X scale
0931     xscaleR = 'lin';
0932     if length(xscales)==2 && strcmpi(xscales{1}, 'all')
0933         xscaleR = xscales{2};
0934     else
0935         if j<=length(xscales)
0936             if ~isempty(xscales{j})
0937                 xscaleR = xscales{j};
0938             end
0939         end
0940     end
0941     set(tsax(end), 'XScale', xscaleR);
0942
0943     % Set X range
0944     if length(xranges) == 2 && strcmpi(xranges{1}, 'all')
0945         set(tsax(end), 'XLim', xrange{2});
0946     else
0947         if j<=length(xranges)
0948             if ~isempty(xranges{j})
0949                 set(tsax(end), 'XLim', xrange{j});
0950             end
0951         end
0952     end
0953
0954     % Set grid on or off
0955     grid(tsax(end), 'on');
0956
0957     %----- line properties
0958
0959     % Set line color
0960     set(li, 'Color', col);
0961
0962     % Set specific colors
0963     if length(linecolors) == 2 && strcmpi(linecolors{1}, 'all')
0964         set(li, 'Color', linecolors{2});
0965     else
0966         if j<=length(linecolors) && ~isempty(linecolors{j})
0967             set(li, 'Color', linecolors{j});
0968         end
0969     end
0970
0971     % Set line style
0972     if length(linestyles) == 2 && strcmpi(linestyles{1}, 'all')
0973         set(li, 'LineStyle', linestyles{2});
0974     else
0975         if j<=length(linestyles) && ~isempty(linestyles{j})
0976             set(li, 'LineStyle', linestyles{j});
0977         end
0978     end
0979
0980     % Set markers
0981     if length(markers) == 2 && strcmpi(markers{1}, 'all')
0982         set(li, 'Marker', markers{2});
0983     else
0984         if j<=length(markers) && ~isempty(markers{j})
0985             set(li, 'Marker', markers{j});
0986         end
0987     end
0988
0989     % Set line widths
0990     if length(linewidths) == 2 && strcmpi(linewidths{1}, 'all')
0991         set(li, 'LineWidth', linewidths{2});
0992     else
0993         if j<=length(linewidths) && ~isempty(linewidths{j})
0994             set(li, 'LineWidth', linewidths{j});
0995         end
0996     end
0997
0998     % Set legend string
0999     lstr = '';
1000     if legendsOn
1001         if j<=length(legends) && ~isempty(legends{j})
1002             lstr = legends{j};
1003         else
1004             lstr = utils.plottools.label(aos(j).name);
1005         end
1006     end
1007     legendStr = [legendStr cellstr(lstr)];
1008
1009     % Set the legend now if we can
1010     if legendsOn
1011         if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
1012             legend(fixlabel(legendStr{end}), 'Location', legendLoc);
1013         end

```

```

1014     end
1015     end
1016
1017     % Process legends for stacked plots
1018     if legendsOn
1019         if strcmp(arrangement, 'stacked')
1020             h = legend(fixlabel(legendStr), 'Location', legendLoc);
1021             set(h, 'FontSize', 10)
1022         end
1023     end
1024 end % End if empty AOs
1025
1026 % Set outputs
1027 if nargout > 0
1028     varargout{1} = tsfig;
1029 end
1030 if nargout > 1
1031     varargout{2} = tsax;
1032 end
1033 if nargout == 3
1034     varargout{3} = tsli;
1035 end
1036 if nargout > 3
1037     error('### Too many output arguments');
1038 end
1039 end % end xy_plot
1040
1041
1042 %-----
1043 % Plot cdata objects
1044 %
1045 function varargout = y_plot(varargin)
1046
1047     aos = varargin{1};
1048     pl = varargin{2};
1049
1050     % Extract parameters
1051     arrangement = find(pl, 'Arrangement');
1052     linecolors = find(pl, 'LineColors');
1053     colors = find(pl, 'Colors');
1054     linestyles = find(pl, 'LineStyles');
1055     markers = find(pl, 'Markers');
1056     linewidths = find(pl, 'LineWidths');
1057     legends = find(pl, 'Legends');
1058     ylabel = find(pl, 'YLabels');
1059     xlabel = find(pl, 'XLabels');
1060     xmaths = find(pl, 'XMaths');
1061     ymaths = find(pl, 'YMaths');
1062     yranges = find(pl, 'YRanges');
1063     xrange = find(pl, 'XRanges');
1064     yscale = find(pl, 'YScales');
1065     xscale = find(pl, 'XScales');
1066     type = find(pl, 'Function');
1067     legendLoc = find(pl, 'LegendLocation');
1068
1069     if ~iscell(xscale), xscale = {xscale}; end
1070     if ~iscell(yscale), yscale = {yscale}; end
1071
1072     % check whether we want legends or not
1073     if iscell(legends)
1074         legendsOn = 1;
1075     else
1076         if strcmp(legends, 'off')
1077             legendsOn = 0;
1078         else
1079             legendsOn = 1;
1080             legends = [];
1081         end
1082     end
1083
1084     if ~iscell(linewidths), linewidths = {linewidths}; end
1085     if ~iscell(linestyles), linestyles = {linestyles}; end
1086     if ~iscell(linecolors), linecolors = {linecolors}; end
1087     if ~iscell(markers), markers = {markers}; end
1088     if ~iscell(legends), legends = {legends}; end
1089     if ~iscell(ylabel), ylabel = {ylabel}; end
1090     if ~iscell(xlabel), xlabel = {xlabel}; end
1091     if ~iscell(xmaths), xmaths = {xmaths}; end
1092     if ~iscell(ymaths), ymaths = {ymaths}; end
1093     if ~iscell(xrange), xrange = {xrange}; end
1094     if ~iscell(yrange), yrange = {yrange}; end
1095
1096     % collect figure handles
1097     tsfig = []; tsax = []; tsli = [];
1098     % Legend holder
1099     legendStr = [];

```

```

1100 if ~isempty(aos)
1101 % Now loop over AOs
1102 Na = length(aos);
1103 for j=1:Na
1104 % what figures do we need?
1105 switch arrangement
1106 case 'single'
1107     tsfig = [tsfig figure];
1108     tsax = subplot(1,1,1);
1109     col = colors{1};
1110 case 'stacked'
1111     if j==1, tsfig = figure; end
1112     tsax = subplot(1,1,1);
1113     col = colors{mod(j-1,length(colors))+1};
1114     hold on;
1115 case 'subplots'
1116     if j == 1, tsfig = figure; end
1117     tsax = [tsax subplot(Na, 1, j)];
1118     col = colors{1};
1119 otherwise
1120     error('### Unknown plot arrangement');
1121 end
1122
1123 %----- Apply math functions
1124 if isreal(aos(j).data.y)
1125     x = 1:length(aos(j).data.y);
1126     y = aos(j).data.y;
1127 else
1128     x = real(aos(j).data.y);
1129     y = imag(aos(j).data.y);
1130 end
1131
1132 ymath = '';
1133 if j<=length(ymaths)
1134     if ~isempty(ymaths{j})
1135         eval(sprintf('y = %s;', ymaths{j}));
1136         ymath = ymaths{j};
1137     end
1138 end
1139 xmath = '';
1140 if j<=length(xmaths)
1141     if ~isempty(xmaths{j})
1142         eval(sprintf('x = %s;', xmaths{j}));
1143         xmath = xmaths{j};
1144     end
1145 end
1146
1147 %----- Plot the data
1148 idcs = feval(type, tsax(end), x, y);
1149 tsli = [tsli idcs(1:end).'];
1150
1151 %----- Axis properties
1152
1153 % Set ylabel
1154 if j<=length(ylabels) && ~isempty(ylabels{j})
1155     ylstr = ylabels{j};
1156 else
1157     ylstr = find(pl, 'YLabel');
1158 end
1159 if ~isempty(ymath)
1160     ymath = strrep(ymath, 'y', sprintf('%s', char(aos(j).data.yunits)));
1161     ylstr = [ylstr ' ' ymath ];
1162 else
1163     ylstr = [ylstr ' ' char(aos(j).data.yunits) ];
1164 end
1165 ylstr = fixlabel(ylstr);
1166 ylabel(ylstr);
1167
1168 % Set xlabel
1169 if j<=length(xlabels) && ~isempty(xlabels{j})
1170     xlstr = xlabels{j};
1171 else
1172     xlstr = find(pl, 'XLabel');
1173 end
1174 if ~isempty(xmath)
1175     xmath = strrep(xmath, 'x', 'N');
1176     xlstr = [xlstr ' ' xmath ];
1177 else
1178     xlstr = [xlstr ' [N] ' ];
1179 end
1180 xlstr = fixlabel(xlstr);
1181 xlabel(xlstr);
1182
1183 % Set Y scale
1184 yscaleR = 'lin';

```



```

1186     if length(yscales)==2 && strcmpi(yscales{1}, 'all')
1187         yscaleR = yscales{2};
1188     else
1189         if j<=length(yscales)
1190             if ~isempty(yscales{j})
1191                 yscaleR = yscales{j};
1192             end
1193         end
1194     end
1195     set(tsax(end), 'YScale', yscaleR);
1196
1197     % Set X scale
1198     xscaleR = 'lin';
1199     if length(xscales)==2 && strcmpi(xscales{1}, 'all')
1200         xscaleR = xscales{2};
1201     else
1202         if j<=length(xscales)
1203             if ~isempty(xscales{j})
1204                 xscaleR = xscales{j};
1205             end
1206         end
1207     end
1208     set(tsax(end), 'XScale', xscaleR);
1209
1210     % Set Y range
1211     if length(yranges) == 2 && strcmpi(yranges{1}, 'all')
1212         set(tsax(end), 'YLim', yranges{2});
1213     else
1214         if j<=length(yranges)
1215             if ~isempty(yranges{j})
1216                 set(tsax(end), 'YLim', yranges{j});
1217             end
1218         end
1219     end
1220
1221     % Set X range
1222     if length(xranges) == 2 && strcmpi(xranges{1}, 'all')
1223         set(tsax(end), 'XLim', xrange{2});
1224     else
1225         if j<=length(xranges)
1226             if ~isempty(xranges{j})
1227                 set(tsax(end), 'XLim', xrange{j});
1228             end
1229         end
1230     end
1231
1232     % Set grid on or off
1233     grid(tsax(end), 'on');
1234
1235     %----- line properties
1236
1237     % Set line color
1238     set(idcs, 'Color', col);
1239
1240     % Override line colors with user defined colors
1241     if length(linecolors) == 2 && strcmpi(linecolors{1}, 'all')
1242         set(idcs, 'Color', linecolors{2});
1243     else
1244         if j<=length(linecolors) && ~isempty(linecolors{j})
1245             set(idcs, 'Color', linecolors{j});
1246         end
1247     end
1248
1249     % Set line style
1250     if length(linestyles) == 2 && strcmpi(linestyles{1}, 'all')
1251         set(idcs, 'LineStyle', linestyles{2});
1252     else
1253         if j<=length(linestyles) && ~isempty(linestyles{j})
1254             set(idcs, 'LineStyle', linestyles{j});
1255         end
1256     end
1257
1258     % Set Markers
1259     if length(markers) == 2 && strcmpi(markers{1}, 'all')
1260         set(idcs, 'Marker', markers{2});
1261     else
1262         if j<=length(markers) && ~isempty(markers{j})
1263             set(idcs, 'Marker', markers{j});
1264         end
1265     end
1266
1267     % Set line widths
1268     if length(linewidths) == 2 && strcmpi(linewidths{1}, 'all')
1269         set(idcs, 'LineWidth', linewidths{2});
1270     else
1271         if j<=length(linewidths) && ~isempty(linewidths{j})

```

```

1272         set(idcs, 'LineWidth', linewidths{j});
1273     end
1274 end
1275
1276 % Set legend string
1277 if legendsOn
1278     if j<=length(legends) && ~isempty(legends{j})
1279         legendStr = [legendStr legends{j}];
1280     else
1281         lstr = utils.plottools.label(aos(j).name);
1282         legendStr = [legendStr cellstr(lstr)];
1283     end
1284 end
1285
1286 % Set the legend now if we can
1287 if legendsOn
1288     if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
1289         legend(fixlabel(legendStr{end}), 'Location', legendLoc);
1290     end
1291 end
1292
1293 % Process legends for stacked plots
1294 if legendsOn
1295     if strcmp(arrangement, 'stacked')
1296         h = legend(fixlabel(legendStr), 'Location', legendLoc);
1297         set(h, 'FontSize', 10)
1298     end
1299 end
1300
1301 end
1302
1303 % Set outputs
1304 if nargout > 0
1305     varargout{1} = tsfig;
1306 end
1307 if nargout > 1
1308     varargout{2} = tsax;
1309 end
1310 if nargout == 3
1311     varargout{3} = tsli;
1312 end
1313 if nargout > 3
1314     error('### Too many output arguments');
1315 end
1316 end % End y_plot
1317
1318 %-----
1319 % Plot xyzdata objects
1320 %
1321 %
1322 function varargout = xyz_plot(varargin)
1323
1324     aos = varargin{1};
1325     pl = varargin{2};
1326
1327 % Extract parameters
1328 arrangement = find(pl, 'Arrangement');
1329 linecolors = find(pl, 'LineColors');
1330 colors = find(pl, 'Colors');
1331 linestyle = find(pl, 'LineStyle');
1332 linewidths = find(pl, 'LineWidths');
1333 legends = find(pl, 'Legends');
1334 zlabels = find(pl, 'ZLabels');
1335 ylabels = find(pl, 'YLabels');
1336 xlabels = find(pl, 'XLabels');
1337 xmaths = find(pl, 'XMaths');
1338 ymaths = find(pl, 'YMaths');
1339 zmaths = find(pl, 'ZMaths');
1340 legendLoc = find(pl, 'LegendLocation');
1341
1342 % check whether we want legends or not
1343 if iscell(legends)
1344     legendsOn = 1;
1345 else
1346     if strcmp(legends, 'off')
1347         legendsOn = 0;
1348     else
1349         legendsOn = 1;
1350     end
1351 end
1352
1353 % collect figure handles
1354 tsfig = [];
1355 tsax = [];
1356 tsli = [];

```



```

1358
1359 % Legend holder
1360 legendStr = [];
1361
1362 if ~isempty(aos)
1363
1364     % Now loop over AOs
1365     Na = length(aos);
1366     for j=1:Na
1367         % what figures do we need?
1368         switch arrangement
1369             case 'single'
1370                 tsfig = [tsfig figure];
1371                 tsax = subplot(1,1,1);
1372             case 'subplots'
1373                 if j == 1, tsfig = figure; end
1374                 tsax = [tsax subplot(Na, 1, j)];
1375             otherwise
1376                 error('### Unknown plot arrangement');
1377         end
1378
1379         %----- Apply math functions
1380         x = aos(j).data.x;
1381         y = aos(j).data.y;
1382         z = aos(j).data.z;
1383
1384         ymath = '';
1385         if j<=length(ymaths)
1386             if ~isempty(ymaths{j})
1387                 eval(sprintf('y = %s;', ymaths{j}));
1388                 ymath = ymaths{j};
1389             end
1390         end
1391         xmath = '';
1392         if j<=length(xmaths)
1393             if ~isempty(xmaths{j})
1394                 eval(sprintf('x = %s;', xmaths{j}));
1395                 xmath = xmaths{j};
1396             end
1397         end
1398         zmath = '';
1399         if j<=length(zmaths)
1400             if ~isempty(zmaths{j})
1401                 eval(sprintf('z = %s;', zmaths{j}));
1402                 zmath = zmaths{j};
1403             end
1404         end
1405
1406         %----- Plot the data
1407
1408         idcs = pcolor(x,y,z);
1409         tsli = [tsli idcs(1:end).'];
1410
1411         % plot properties
1412         set(idcs, 'EdgeColor', 'none');
1413
1414         %----- Axis properties
1415
1416         % Reverse y-direction for spectrograms
1417         set(tsax(end), 'YDir', 'reverse');
1418
1419         % Set ylabel
1420         if j<=length(ylabels) && ~isempty(ylabels{j})
1421             ylstr = ylabels{j};
1422         else
1423             ylstr = find(pl, 'YLabel');
1424         end
1425         if ~isempty(ymath)
1426             ymath = strrep(ymath, 'y', sprintf('%s', char(aos(j).data.yunits)));
1427             ylstr = [ylstr ' ' ymath ];
1428         else
1429             ylstr = [ylstr ' ' char(aos(j).data.yunits) ];
1430         end
1431         ylstr = fixlabel(ylstr);
1432         ylabel(ylstr);
1433
1434         % Set xlabel
1435         if j<=length(xlabels) && ~isempty(xlabels{j})
1436             xlstr = xlabels{j};
1437         else
1438             xlstr = find(pl, 'XLabel');
1439         end
1440         if ~isempty(xmath)
1441             xmath = strrep(xmath, 'x', sprintf('%s', char(aos(j).data.xunits)));
1442             xlstr = [xlstr ' ' xmath ];
1443         else

```

```

1444     xlstr = [xlstr ' ' char(aos(j).data.xunits) ];
1445 end
1446 xlstr = fixlabel(xlstr);
1447 xlabel(xlstr);
1448
1449 % Set grid on or off
1450 grid(tsax(end), 'on');
1451
1452 % Set title string
1453 if ~strcmpi(legends, 'off')
1454     if j<=length(legends) && ~isempty(legends{j})
1455         legendStr = [legendStr legends{j}];
1456     else
1457         lstr = utils.plottools.label(aos(j).name);
1458         legendStr = [legendStr cellstr(lstr)];
1459     end
1460 end
1461
1462 % Set the legend now if we can
1463 tstr = legendStr{end};
1464 if legendsOn
1465     title(tstr);
1466 end
1467
1468 % Set colorbars
1469 hc = colorbar('peer', tsax(end));
1470 if j<=length(zlabels)
1471     if ~isempty(zlabels{j})
1472         zlstr = zlabels{j};
1473     end
1474 else
1475     zlstr = find(pl, 'Zlabel');
1476 end
1477 if ~isempty(zmath), zlstr = [zlstr sprintf('\n%s', zmath)]; end
1478 ylh = get(hc, 'YLabel');
1479 set(ylh, 'String', zlstr);
1480 set(ylh, 'FontSize', get(tsax(end), 'FontSize'))
1481 set(ylh, 'FontName', get(tsax(end), 'FontName'))
1482 set(ylh, 'FontAngle', get(tsax(end), 'FontAngle'))
1483 set(ylh, 'FontWeight', get(tsax(end), 'FontWeight'))
1484 end
1485 end
1486
1487 % Set outputs
1488 if nargout > 0
1489     varargout{1} = tsfig;
1490 end
1491 if nargout > 1
1492     varargout{2} = tsax;
1493 end
1494 if nargout == 3
1495     varargout{3} = tsli;
1496 end
1497 if nargout > 3
1498     error('### Too many output arguments');
1499 end
1500 end % end xyz_plot
1501
1502 %-----
1503 % Get Info Object
1504 %-----
1505 function ii = getInfo(varargin)
1506     if nargin == 1 && strcmpi(varargin{1}, 'None')
1507         sets = {};
1508         pl = [];
1509     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
1510         sets{1} = varargin{1};
1511         pl = getDefaultPlist(sets{1});
1512     else
1513         sets = {'Time-series plot', 'Frequency-series plot', 'Y data plot', 'X-Y data
plot', '3D plot'};
1514         % get plists
1515         pl(size(sets)) = plist;
1516         for k = 1:numel(sets)
1517             pl(k) = getDefaultPlist(sets{k});
1518         end
1519     end
1520     % Build info object
1521     ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: iplot.m,v 1.48
2008/09/05 14:14:33 hewitson Exp $', sets, pl);
1522 end
1523
1524 %-----
1525 % Get Default Plist
1526 %-----
1527 function out = getDefaultPlist(set)

```

```

1528
1529 % Get the LTPDA color set for lines
1530 colors = getappdata(0, 'ltpda_default_plot_colors');
1531
1532 out = plist(...
1533     'Colors', colors, ...
1534     'Arrangement', 'stacked', ...
1535     'Function', 'plot', 'LegendLocation', 'Best');
1536
1537 switch set
1538 case 'Frequency-series plot'
1539     out.append(...
1540         'complexPlotType', 'absdeg', ...
1541         'XLabel', 'Frequency');
1542 case 'Time-series plot'
1543     out.append(...
1544         'Xlabel', 'Time', ...
1545         'Ylabel', 'Amplitude');
1546 case 'X-Y data plot'
1547     out.append(...
1548         'Xlabel', 'X-data', ...
1549         'Ylabel', 'Y-data', ...
1550         'YMaths', '}', ...
1551         'XMaths', '{');
1552 case '3D plot'
1553     out = plist(...
1554         'Colors', colors, ...
1555         'Arrangement', 'single', ...
1556         'Xlabel', 'Time', ...
1557         'Ylabel', 'Frequency', ...
1558         'Zlabel', 'Amplitude', ...
1559         'YMaths', '}', ...
1560         'ZMaths', '}', ...
1561         'XMaths', '{');
1562 case 'Y data plot'
1563     out.append(...
1564         'Xlabel', 'Index', ...
1565         'Ylabel', 'Value');
1566 otherwise
1567     out = plist();
1568 end
1569 end
1570
1571 % Perform some substitutions on the labels
1572 function ss = fixlabel(ss)
1573
1574 wasCell = true;
1575 if ~iscell(ss)
1576     ss = {ss};
1577     wasCell = false;
1578 end
1579
1580 for kk=1:numel(ss)
1581     s = ss{kk};
1582
1583     % Replace all ^(...) with ^{...}
1584     j = 1;
1585     while j<numel(s)
1586         if strcmp(s(j:j+1), '^(')
1587             % find next )
1588             for k=1:numel(s)-j+1
1589                 if s(j+k) == ')'
1590                     s(j+1) = '{';
1591                     s(j+k) = '}';
1592                     break;
1593                 end
1594             end
1595         end
1596         j = j + 1;
1597     end
1598     % Replace all .^ with ^
1599     s = strrep(s, '.^', '^');
1600
1601     ss(kk) = {s};
1602 end
1603
1604 if ~wasCell
1605     ss = ss{1};
1606 end
1607
1608 end
1609

```

plot

PURPOSE ^

PLOT a simple plot of analysis objects.

SYNOPSIS ^

function varargout = plot(varargin)

DESCRIPTION ^

```
PLOT a simple plot of analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PLOT a simple plot of analysis objects.

CALL:
    plot(ao)
    plot(ao,LineStyle)
    plot(axes_handle, ao)
    plot(axes_handle, ao, LineSpec)
    plot(...,'PropertyName',PropertyValue,...)
    [line_h] = plot(...)
    [line_h, axes_h] = plot(...)
    [line_h, axes_h, figure_h] = plot(...)

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('plot')

    Get information about a specified set-plist by calling:
    >> ao.getInfo('plot', 'None')

VERSION:      $Id: plot.m,v 1.36 2008/09/05 11:05:29 ingo Exp $

HISTORY:      16-08-2007 Diepholz
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [imag](#) IMAG overloads the imaginary operator for Analysis objects.
-  [plot](#) PLOT a simple plot of analysis objects.
-  [real](#) REAL overloads the real operator for Analysis objects.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [plot](#) PLOT a simple plot of analysis objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % PLOT a simple plot of analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PLOT a simple plot of analysis objects.
0005 %
0006 % CALL:      plot(ao)
0007 %            plot(ao,LineStyle)
0008 %            plot(axes_handle, ao)
0009 %            plot(axes_handle, ao, LineSpec)
0010 %            plot(...,'PropertyName',PropertyValue,...)
0011 %            [line_h] = plot(...)
0012 %            [line_h, axes_h] = plot(...)
0013 %            [line_h, axes_h, figure_h] = plot(...)
0014 %
0015 % M-FILE INFO: Get information about this methods by calling
0016 %              >> ao.getInfo('plot')
0017 %
0018 %              Get information about a specified set-plist by calling:
0019 %              >> ao.getInfo('plot', 'None')
0020 %
0021 % VERSION:    $Id: plot.m,v 1.36 2008/09/05 11:05:29 ingo Exp $
0022 %
0023 % HISTORY:    16-08-2007 Diepholz
0024 %              Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 function varargout = plot(varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 import utils.const.*
0037 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0038
0039 axes_handle = [];
0040 ao_s        = [];
0041 line_spec   = [];
0042 prop_val    = {};
0043 lines_h     = [];
0044
0045 %%%%%%%%%%% Check input values %%%%%%%%%%%
0046 var_argin = varargin;
0047
0048 while ~isempty(var_argin)
0049     if isempty(var_argin{1})
0050         var_argin = var_argin(2:end);
0051     elseif ishandle(var_argin{1}) && isempty(axes_handle)
0052         axes_handle = var_argin{1};
0053         var_argin = var_argin(2:end);
0054     elseif isa(var_argin{1}, 'ao')
0055         var_argin{1} = reshape(var_argin{1}, 1, []);
0056         ao_s = [ao_s var_argin{1}];
0057         var_argin = var_argin(2:end);
0058     elseif ischar(var_argin{1}) && islinespec(var_argin{1}) && isempty(line_spec)
0059         line_spec = var_argin{1};
0060         var_argin = var_argin(2:end);
0061     else
0062         if length(var_argin) == 1
0063             error('### There is only one property [%s] without value left', var_argin{1});
0064         elseif length(var_argin) >= 2
0065             prop = var_argin{1};
0066             val = var_argin{2};
0067             prop_val = [prop_val, prop, val];
0068             var_argin = var_argin(3:end);
0069         end
0070     end
0071 end
0072
0073 %%%%%%%%%%% For all ao_s %%%%%%%%%%%
0074 %
0075 %%%%%%%%%%% Set the line colors %%%%%%%%%%%
0076 for ii = 1:length(ao_s)
0077
0078     %%%%%%%%%%% Set the line colors %%%%%%%%%%%
0079     line_colors = getappdata(0, 'ltpda_default_plot_colors');
0080
0081     %% Set only the color if the color is not set in the linespec
0082     [res, style] = islinespec(line_spec);
0083
0084     if isempty(style{3})
0085         prop_val{end+1} = 'Color';
0086         prop_val{end+1} = line_colors{ii};

```

```

0087     end
0088
0089     %%%%%%%%%%% Get the x-, y-values %%%%%%%%%%%
0090     x = ao_s(ii).data.getX;
0091     y = ao_s(ii).data.y;
0092
0093     if isreal(y)
0094         if isempty(x)
0095             x = 1:length(y);
0096         end
0097     else % iscomplex(y)
0098         if isempty(x)
0099             x = real(y);
0100             y = imag(y);
0101         end
0102     end
0103
0104     %%%%%%%%%%% Call the different plot command lines %%%%%%%%%%%
0105     if ~isempty(axes_handle) && ~isempty(line_spec)
0106         line_h = plot(axes_handle, x, y, line_spec);
0107     elseif ~isempty(axes_handle)
0108         line_h = plot(axes_handle, x, y);
0109     elseif ~isempty(line_spec)
0110         line_h = plot(x, y, line_spec);
0111     else
0112         line_h = plot(x, y);
0113     end
0114
0115     hold on;
0116
0117     line_h = reshape(line_h, 1, []);
0118     lines_h = [lines_h line_h];
0119
0120     %%%%%%%%%%% Set the different handles: axes- and figure- handle %%%%%%%%%%%
0121     axes_h = get(line_h(1), 'Parent');
0122     figure_h = get(axes_h, 'Parent');
0123
0124     %%%%%%%%%%% Set line series Properties OR axes Properties %%%%%%%%%%%
0125     while length(prop_val) >= 2
0126         prop = prop_val{1};
0127         val = prop_val{2};
0128         prop_val = prop_val(3:end);
0129         if isprop(line_h, prop)
0130             set(line_h, prop, val);
0131         else
0132             if isprop(axes_h, prop)
0133                 set(axes_h, prop, val);
0134             else
0135                 error('### Invalid line property OR axes property: '%s'.', prop);
0136             end
0137         end
0138     end
0139
0140     end % for ii = 1:length(ao_s)
0141
0142
0143     %%%%%%%%%%% varargout{1} = line handle %%%%%%%%%%%
0144     %%%%%%%%%%% varargout{2} = axes handle %%%%%%%%%%%
0145     %%%%%%%%%%% varargout{3} = figure handle %%%%%%%%%%%
0146     if nargout == 0
0147     elseif nargout == 1
0148         varargout{1} = lines_h;
0149     elseif nargout == 2
0150         varargout{1} = lines_h;
0151         varargout{2} = axes_h;
0152     elseif nargout == 3
0153         varargout{1} = lines_h;
0154         varargout{2} = axes_h;
0155         varargout{3} = figure_h;
0156     else
0157         error('### Unknown number of outputs.')
0158     end
0159
0160     hold off;
0161 end
0162
0163 %-----
0164 % Get Info Object
0165 %-----
0166
0167 function ii = getInfo(varargin)
0168     if nargin == 1 && strcmpi(varargin{1}, 'None')
0169         sets = {};
0170         pl = [];
0171     else
0172         sets = {'Default'};

```

```
0173     pl    = getDefaultPlist;  
0174     end  
0175     % Build info object  
0176     ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: plot.m,v 1.36  
2008/09/05 11:05:29 ingo Exp $', sets, pl);  
0177     end  
0178  
0179     %-----  
0180     % Get Default Plist  
0181     %-----  
0182     function pl = getDefaultPlist()  
0183         pl = plist();  
0184     end  
0185  
0186     % END  
0187  
0188
```

save

PURPOSE ^

SAVE overloads save operator for ltpda objects.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
SAVE overloads save operator for ltpda objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SAVE overloads save operator for ltpda objects.

CALL:      save(a, 'blah.mat') Save an object as a .mat file.
           save(a, 'blah.xml') Save an object as an XML file.

PARAMETERS: 'filename' - name of the file

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('save')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('save', 'set')

VERSION:    $Id: save.m,v 1.5 2008/09/07 18:15:04 hewitson Exp $

HISTORY:    14-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [save](#) SAVE overloads save operator for ltpda objects.

This function is called by:

-  [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.
-  [save](#) SAVE overloads save operator for ltpda objects.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % SAVE overloads save operator for ltpda objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SAVE overloads save operator for ltpda objects.
0005 %
0006 % CALL:      save(a, 'blah.mat') Save an object as a .mat file.
0007 %           save(a, 'blah.xml') Save an object as an XML file.
0008 %
0009 % PARAMETERS: 'filename' - name of the file
```



```

0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 % >> ao.getInfo('save')
0013 %
0014 % Get information about a specified set-plist by calling:
0015 % >> ao.getInfo('save', 'set')
0016 %
0017 % VERSION: $Id: save.m,v 1.5 2008/09/07 18:15:04 hewitson Exp $
0018 %
0019 % HISTORY: 14-02-07 M Hewitson
0020 % Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = save(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 %%% Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names[ii] = inputname(ii);end
0035 %
0036 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), '', in_names);
0037 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist', in_names);
0038 %
0039 %%% REMARK: Special case for the plist-class because collect_objects collects
0040 %%% ALL plist-objects even the plist which should set the property.
0041 %%% In this case must be the plist which sets the property
0042 %%% at the last position.
0043 if isa(objs, 'plist')
0044     if nparams(objs(end)) == 1 && isparam(objs(end), 'filename')
0045         pls = [pls objs(end)];
0046         objs(end) = [];
0047     end
0048 end
0049 %
0050 %%% Combine the plists
0051 if isa(pls, 'plist')
0052     pls = combine(pls);
0053 end
0054 %
0055 %%% If we eliminated the objects and plists then is the rest the property name
0056 if length(rest) == 1 || ~isparam(pls, 'filename')
0057     filename = rest;
0058 else
0059     error('### Please specify [only one] filename.')
0060 end
0061 %
0062 %%% If rest contains the filename then use this filename
0063 if length(filename) == 1 && ischar(filename{1})
0064     pls = combine(plist('filename', filename{1}), pls);
0065 end
0066 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Save object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0067 %
0068 % get filename
0069 filename = find(pls, 'filename');
0070 %
0071 % Inspect filename
0072 [path, name, ext] = fileparts(filename);
0073 %
0074 switch ext
0075 case '.mat'
0076     objs = utils.prog.rstruct(objs);
0077     save(filename, 'objs');
0078 case '.xml'
0079     % convert object to xml
0080     xml = com.mathworks.xml.XMLUtils.createDocument('ltpda_object');
0081     parent = xml.getDocumentElement;
0082     %
0083     % add style sheet
0084     stylesheet = xml.createProcessingInstruction('xml-stylesheet',
0085 'type="text/xsl" href="ingo.xsl"');
0086     xml.insertBefore(stylesheet, xml.getChildNodes.item(0));
0087 %
0088 % add Attribute 'ltpda_version' to the root node
0089 ltpda_version = getappdata(0, 'ltpda_version');
0090 parent.setAttribute('ltpda_version', ltpda_version);
0091 %
0092 %
0093 %
0094 %

```

```
0095     utils.helper.xmlwrite(objs, xml, parent, '');    % Save the XML document.
0096     xmlwrite(filename, xml);
0097
0098     otherwise
0099         error('### unknown file extension.');
```

end

end

0103

0104 %%

0105 %

0106 %%

0107

0108 %%

0109 %

0110 % FUNCTION: getInfo

0111 %

0112 % DESCRIPTION: Get Info Object

0113 %

0114 % HISTORY: 11-07-07 M Hewitson

0115 %

0116 %

0117 %%

0118

0119 function ii = getInfo(varargin)

0120 if nargin == 1 && strcmpi(varargin{1}, 'None')

0121 sets = {};

0122 pl = [];

0123 else

0124 sets = {'Default'};

0125 pl = getDefaultPlist;

0126 end

0127 % Build info object

0128 ii = minfo(mfilename, 'ltpda_uo', '', utils.const.categories.output, '\$Id: save.m,v

1.5 2008/09/07 18:15:04 hewitson Exp \$', sets, pl);

0129 end

0130

0131 %%

0132 %

0133 % FUNCTION: getDefaultPlist

0134 %

0135 % DESCRIPTION: Get Default Plist

0136 %

0137 % HISTORY: 11-07-07 M Hewitson

0138 %

0139 %

0140 %%

0141

0142 function plo = getDefaultPlist()

0143 plo = plist('filename', 'object.xml');

0144 end

0145

type

PURPOSE ^

TYPE converts the input objects to MATLAB functions.

SYNOPSIS ^

function varargout = type(varargin)

DESCRIPTION ^

```
TYPE converts the input objects to MATLAB functions.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: TYPE converts the input objects to MATLAB functions that
              reproduce the processing steps that led to the input objects.

CALL:
      type(as)
      type(as, 'filename')
      type(as, plist)

INPUTS:
      as - array of ltpda_uoh objects
      pl - parameter list (see below)

OUTPUTS:
      none.

PARAMETER LIST:
      'filename' - specify the filename to write the commands in.

M-FILE INFO: Get information about this methods by calling
              >> <ltpda_uoh>.getInfo('type')

              Get information about a specified set-plist by calling:
              >> <ltpda_uoh>.getInfo('type', 'None')

              example: ao.getInfo('type')

VERSION:      $Id: type.m,v 1.5 2008/09/04 15:29:30 ingo Exp $

HISTORY:      02-02-07 M Hewitson
              Created

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % TYPE converts the input objects to MATLAB functions.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TYPE converts the input objects to MATLAB functions that
0005 %              reproduce the processing steps that led to the input objects.
0006 %
```

```

0007 % CALL:      type(as)
0008 %            type(as, 'filename')
0009 %            type(as, plist)
0010 %
0011 % INPUTS:      as - array of ltpda_uoh objects
0012 %              pl - parameter list (see below)
0013 %
0014 % OUTPUTS:      none.
0015 %
0016 % PARAMETER LIST:
0017 %
0018 %              'filename' - specify the filename to write the commands in.
0019 %
0020 % M-FILE INFO:  Get information about this methods by calling
0021 %              >> <ltpda_uoh>.getInfo('type')
0022 %
0023 %              Get information about a specified set-plist by calling:
0024 %              >> <ltpda_uoh>.getInfo('type', 'None')
0025 %
0026 %              example: ao.getInfo('type')
0027 %
0028 % VERSION:      $Id: type.m,v 1.5 2008/09/04 15:29:30 ingo Exp $
0029 %
0030 % HISTORY:      02-02-07 M Hewitson
0031 %              Created
0032 %
0033 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034
0035 function varargout = type(varargin)
0036
0037     import utils.const.*
0038
0039     % Check if this is a call for parameters
0040     if utils.helper.isinfocall(varargin{:})
0041         varargout{1} = getInfo(varargin{3});
0042         return
0043     end
0044
0045     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0046
0047     % Collect input variable names
0048     in_names = cell(size(varargin));
0049     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0050
0051     % Collect all AOs and plists
0052     [as, ao_invars, rest] = utils.helper.collect_objects(varargin(:), 'ltpda_uoh',
in_names);
0053     [pl, pl_invars, rest] = utils.helper.collect_objects(rest, 'plist', in_names);
0054
0055     % Combine plists
0056     pl = combine(pl, getDefaultPlist);
0057     filename = find(pl, 'filename');
0058
0059     % Check for filename
0060     if isempty(filename)
0061         if ~isempty(rest)
0062             if strcmp(rest{1}(end-1:end), '.m')
0063                 filename = rest{1};
0064             else
0065                 filename = fullfile(rest{1}, '.m');
0066             end
0067         end
0068     end
0069
0070     % Go through each input AO
0071     for j=1: numel(as)
0072
0073         % convert to commands
0074         fcn(j).cmds = hist2m(as(j).hist);
0075
0076     end % End AO loop
0077
0078     %----- Output to screen
0079     if isempty(filename)
0080
0081         for j=1: numel(fcn)
0082             % Header
0083             LL = 25;
0084             disp(utils.prog.strpad('', [LL LL], '-'));
0085             disp(utils.prog.strpad(as(j).name, [LL LL], '-'));
0086             disp(utils.prog.strpad('', [LL LL], '-'));
0087             disp(' ');
0088
0089             % Print each command
0090             for k=numel(fcn(j).cmds):-1:1
0091                 disp(fcn(j).cmds{k})

```

```

0092     end
0093
0094     % Footer
0095     disp(' ');
0096     disp( utils.prog.strpad(' ', [LL LL], '-'));
0097     disp( utils.prog.strpad(as(j).name, [LL LL], '-'));
0098     disp( utils.prog.strpad(' ', [LL LL], '-'));
0099     disp(' ');
0100 end
0101
0102 %----- Output to file
0103 else
0104
0105     % clear this m-file from memory
0106     clear(filename);
0107
0108     % Open the file
0109     ii = getInfo;
0110     fd = fopen(filename, 'w+');
0111     [path, name, ext, vers] = fileparts(filename);
0112
0113     % Write help header
0114     fprintf(fd, '%% %s \n', upper(filename));
0115     fprintf(fd, '%% \n');
0116     fprintf(fd, '%% \n');
0117     fprintf(fd, '%% written by %s / %s\n', mfilename, ii.mversion);
0118     fprintf(fd, '%% \n');
0119     fprintf(fd, '%% \n');
0120     fprintf(fd, ' \n');
0121     fprintf(fd, ' \n');
0122
0123     % write main fcn
0124     fprintf(fd, 'function out = %s\n\n', name);
0125     fprintf(fd, 'out = [];\n');
0126     for j=1:numel(as)
0127         fprintf(fd, 'out = [out obj%03d];\n', j);
0128     end
0129     fprintf(fd, '\nend\n\n');
0130
0131     % Write each sub function
0132     for kk=1:numel(fcn)
0133         % header
0134         fprintf(fd, '%% sub-function for object %s\n', as(kk).name);
0135         fprintf(fd, 'function a_out = obj%03d\n\n', kk);
0136         % write command for this function
0137         for jj=numel(fcn(kk).cmds):-1:1
0138             fprintf(fd, '\t%s\n', fcn(kk).cmds{jj});
0139         end
0140         % Footer
0141         fprintf(fd, '\nend\n\n');
0142     end
0143
0144     fprintf(fd, '\n\n%% END\n');
0145
0146     % Close file
0147     fclose(fd);
0148     % This somehow forces the new file to be available to MATLAB
0149     cd ( '.');
0150 end
0151
0152 end
0153
0154 %-----
0155 % Get Info Object
0156 %-----
0157 function ii = getInfo(varargin)
0158     if nargin == 1 && strcmpi(varargin{1}, 'None')
0159         sets = {};
0160         pl = [];
0161     else
0162         sets = {'Default'};
0163         pl = getDefaultPlist;
0164     end
0165     % Build info object
0166     ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: type.m,v 1.5
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0167 end
0168
0169 %-----
0170 % Get Default Plist
0171 %-----
0172 function pl = getDefaultPlist()
0173     pl = plist('filename', '');
0174 end
0175
0176

```

0177
0178 % END

ge

PURPOSE ^

GE overloads >= operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function varargout = ge(varargin)

DESCRIPTION ^

```
GE overloads >= operator for analysis objects. Compare the y-axis values.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: GE overloads >= operator for analysis objects.
             Compare the y-axis values.

CALL:       a = b>=c;

INPUTS:     b - Analysis object
             c - Analysis object OR number

OUTPUTS:    a - a cdata Analysis Object containing logical values from the
             comparison.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('ge')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('ge', 'None')

VERSION:    $Id: ge.m,v 1.14 2008/09/05 11:05:29 ingo Exp $

HISTORY:    06-02-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **x** X Get the data property 'x'.
-  **y** Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl_default = getDefaultPlist()

SOURCE CODE ^

```
0001 % GE overloads >= operator for analysis objects. Compare the y-axis values.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: GE overloads >= operator for analysis objects.
0005 %             Compare the y-axis values.
0006 %
0007 % CALL:       a = b>=c;
0008 %
```

```

0009 % INPUTS:      b - Analysis object
0010 %              c - Analysis object OR number
0011 %
0012 % OUTPUTS:     a - a cdata Analysis Object containing logical values from the
0013 %               comparison.
0014 %
0015 % M-FILE INFO:  Get information about this methods by calling
0016 %               >> ao.getInfo('ge')
0017 %
0018 %               Get information about a specified set-plist by calling:
0019 %               >> ao.getInfo('ge', 'None')
0020 %
0021 % VERSION:      $Id: ge.m,v 1.14 2008/09/05 11:05:29 ingo Exp $
0022 %
0023 % HISTORY:      06-02-2007 M Hewitson
0024 %               Creation
0025 %
0026 %%%%%%%%%%%
0027
0028 function varargout = ge(varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 if ~(nargin == 2 && ...
0037     ((isa(varargin{1}, 'ao') && isa(varargin{2}, 'ao')) || ...
0038     (isa(varargin{1}, 'ao') && isnumeric(varargin{2}))))
0039     error('### comparisons only between analysis objects and/or numbers.')
0040 end
0041
0042 x = varargin{1}.data.getY;
0043
0044 if isa(varargin{2}, 'ao')
0045     y = varargin{2}.data.getY;
0046 else
0047     y = varargin{2}*ones(size(x));
0048 end
0049
0050 a = x >= y;
0051
0052 % Set output
0053 varargout{1} = a;
0054 end
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl = [];
0062     else
0063         sets = {'Default'};
0064         pl = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.relop, '$Id: ge.m,v 1.14
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0068 end
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075
0076

```


gt

PURPOSE ^

GT overloads > operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function varargout = gt(varargin)

DESCRIPTION ^

```
GT overloads > operator for analysis objects. Compare the y-axis values.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: GT overloads > operator for analysis objects.
              Compare the y-axis values.

CALL:        a = b>c;

INPUTS:      b - Analysis object
              c - Analysis object OR number

OUTPUTS:     a - a cdata Analysis Object containing logical values from the
              comparison.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('gt')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('gt', 'None')

VERSION:     $Id: gt.m,v 1.14 2008/09/05 11:05:29 ingo Exp $

HISTORY:     06-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GT overloads > operator for analysis objects. Compare the y-axis values.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: GT overloads > operator for analysis objects.
0005 %               Compare the y-axis values.
0006 %
0007 % CALL:        a = b>c;
0008 %
```

```

0009 % INPUTS:      b - Analysis object
0010 %              c - Analysis object OR number
0011 %
0012 % OUTPUTS:     a - a cdata Analysis Object containing logical values from the
0013 %               comparison.
0014 %
0015 % M-FILE INFO:  Get information about this methods by calling
0016 %               >> ao.getInfo('gt')
0017 %
0018 %               Get information about a specified set-plist by calling:
0019 %               >> ao.getInfo('gt', 'None')
0020 %
0021 % VERSION:      $Id: gt.m,v 1.14 2008/09/05 11:05:29 ingo Exp $
0022 %
0023 % HISTORY:      06-02-2007 M Hewitson
0024 %               Creation
0025 %
0026 %%%%%%%%%%%
0027
0028 function varargout = gt(varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 if ~(nargin == 2 && ...
0037     ((isa(varargin{1}, 'ao') && isa(varargin{2}, 'ao')) || ...
0038     (isa(varargin{1}, 'ao') && isnumeric(varargin{2}))))
0039     error('### comparisons only between analysis objects and/or numbers.')
0040 end
0041
0042 x = varargin{1}.data.getY;
0043
0044 if isa(varargin{2}, 'ao')
0045     y = varargin{2}.data.getY;
0046 else
0047     y = varargin{2}*ones(size(x));
0048 end
0049
0050 a = x > y;
0051
0052 % Set output
0053 varargout{1} = a;
0054 end
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl = [];
0062     else
0063         sets = {'Default'};
0064         pl = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.relop, '$Id: gt.m,v 1.14
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0068 end
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075
0076

```

le

PURPOSE ^

LE overloads <= operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function varargout = le(varargin)

DESCRIPTION ^

LE overloads <= operator for analysis objects. Compare the y-axis values.
DESCRIPTION: LE overloads <= operator for analysis objects.
Compare the y-axis values.
CALL: a = b<=c;
INPUTS: b - Analysis object
c - Analysis object Or a number
OUTPUTS: a - a cdata Analysis Object containing logical values from the
comparison.
M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('le')
Get information about a specified set-plist by calling:
>> ao.getInfo('le', 'None')
VERSION: \$Id: le.m,v 1.15 2008/09/05 11:05:29 ingo Exp \$
HISTORY: 06-02-2007 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

- x X Get the data property 'x'.
- y Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

- function ii = getInfo(varargin)
- function pl_default = getDefaultPlist()

SOURCE CODE ^

0001 % LE overloads <= operator for analysis objects. Compare the y-axis values.
0002 %
0003 %
0004 % DESCRIPTION: LE overloads <= operator for analysis objects.
0005 % Compare the y-axis values.
0006 %
0007 % CALL: a = b<=c;

```

0008 %
0009 % INPUTS:      b - Analysis object
0010 %              c - Analysis object Or a number
0011 %
0012 % OUTPUTS:
0013 %              a - a cdata Analysis Object containing logical values from the
0014 %                  comparison.
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 %              >> ao.getInfo('le')
0018 %
0019 %              Get information about a specified set-plist by calling:
0020 %              >> ao.getInfo('le', 'None')
0021 %
0022 % VERSION:      $Id: le.m,v 1.15 2008/09/05 11:05:29 ingo Exp $
0023 %
0024 % HISTORY:      06-02-2007 M Hewitson
0025 %                  Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = le(varargin)
0030
0031 % Check if this is a call for parameters
0032 if utils.helper.isinfocall(varargin{:})
0033     varargout{1} = getInfo(varargin{3});
0034     return
0035 end
0036
0037 if ~(nargin == 2 && ...
0038     ((isa(varargin{1}, 'ao') && isa(varargin{2}, 'ao')) || ...
0039     (isa(varargin{1}, 'ao') && isnumeric(varargin{2}))))
0040     error('### comparisons only between analysis objects and/or numbers.')
0041 end
0042
0043 x = varargin{1}.data.getY;
0044
0045 if isa(varargin{2}, 'ao')
0046     y = varargin{2}.data.getY;
0047 else
0048     y = varargin{2}*ones(size(x));
0049 end
0050
0051 a = x <= y;
0052
0053 % Set output
0054 varargout{1} = a;
0055 end
0056
0057 %-----
0058 % Get Info Object
0059 %-----
0060 function ii = getInfo(varargin)
0061     if nargin == 1 && strcmpi(varargin{1}, 'None')
0062         sets = {};
0063         pl    = [];
0064     else
0065         sets = {'Default'};
0066         pl    = getDefaultPlist;
0067     end
0068     % Build info object
0069     ii = minfo(mfilename, 'ao', '', utils.const.categories.relop, '$Id: le.m,v 1.15
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0070 end
0071
0072 %-----
0073 % Get Default Plist
0074 %-----
0075 function pl_default = getDefaultPlist()
0076     pl_default = plist();
0077 end
0078

```

lt

PURPOSE ^

LT overloads < operator for analysis objects. Compare the y-axis values.

SYNOPSIS ^

function varargout = lt(varargin)

DESCRIPTION ^

LT overloads < operator for analysis objects. Compare the y-axis values.
%%%

DESCRIPTION: LT overloads < operator for analysis objects.
Compare the y-axis values.

CALL: a = b<c;

INPUTS: b - Analysis object
c - Analysis object OR number

OUTPUTS: a - a cdata Analysis Object containing logical values from the
comparison.

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('lt')

Get information about a specified set-plist by calling:
>> ao.getInfo('lt', 'None')

VERSION: \$Id: lt.m,v 1.15 2008/09/05 11:05:29 ingo Exp \$

HISTORY: 06-02-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % LT overloads < operator for analysis objects. Compare the y-axis values.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LT overloads < operator for analysis objects.
0005 %               Compare the y-axis values.
0006 %
0007 % CALL:        a = b<c;
0008 %
```

```

0009 % INPUTS:      b - Analysis object
0010 %              c - Analysis object OR number
0011 %
0012 % OUTPUTS:     a - a cdata Analysis Object containing logical values from the
0013 %               comparison.
0014 %
0015 % M-FILE INFO:  Get information about this methods by calling
0016 %               >> ao.getInfo('lt')
0017 %
0018 %               Get information about a specified set-plist by calling:
0019 %               >> ao.getInfo('lt', 'None')
0020 %
0021 % VERSION:      $Id: lt.m,v 1.15 2008/09/05 11:05:29 ingo Exp $
0022 %
0023 % HISTORY:      06-02-2007 M Hewitson
0024 %               Creation
0025 %
0026 %%%%%%%%%%%
0027
0028 function varargout = lt(varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 if ~(nargin == 2 && ...
0037     ((isa(varargin{1}, 'ao') && isa(varargin{2}, 'ao')) || ...
0038     (isa(varargin{1}, 'ao') && isnumeric(varargin{2}))))
0039     error('### comparisons only between analysis objects and/or numbers.')
0040 end
0041
0042 x = varargin{1}.data.getY;
0043
0044 if isa(varargin{2}, 'ao')
0045     y = varargin{2}.data.getY;
0046 else
0047     y = varargin{2}*ones(size(x));
0048 end
0049
0050 a = x < y;
0051
0052 % Set output
0053 varargout{1} = a;
0054 end
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl = [];
0062     else
0063         sets = {'Default'};
0064         pl = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.relop, '$Id: lt.m,v 1.15
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0068 end
0069 %-----
0070 % Get Default Plist
0071 %-----
0072 function pl_default = getDefaultPlist()
0073     pl_default = plist();
0074 end
0075
0076

```

eq

PURPOSE ^

EQ overloads the == operator for ltpda objects.

SYNOPSIS ^

function result = eq(obj1, obj2, varargin)

DESCRIPTION ^

```
EQ overloads the == operator for ltpda objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: EQ overloads the == operator for ltpda objects.

              All fields are checked.

CALL:         result = eq(obj1,obj2)
              result = eq(obj1,obj2,  exc_list)
              result = eq(obj1,obj2, 'property1', 'property2')
              result = eq(obj1,obj2, '<class>/property', '<class>/property')

EXAMPLES:     result = eq(obj1,obj2, 'name', 'created')
              result = eq(obj1,obj2, '<class>/name')

INPUTS:       obj1, obj2 - Input objects
              exc_list  - Exception list
                      List of properties which are not checked.

OUTPUTS:      If the two objects are considered equal, result == 1,
              otherwise, result == 0.

M-FILE INFO:  Get information about this methods by calling
              >> ao.getInfo('eq')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('eq', 'set')

VERSION:      $Id: eq.m,v 1.8 2008/09/04 15:29:30 ingo Exp $

HISTORY:      18-02-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [eq](#) EQ overloads the == operator for ltpda objects.

This function is called by:

-  [eq](#) EQ overloads the == operator for ltpda objects.
-  [ltpda_obj](#) LTPDA_OBJ is the abstract ltpda base class.
-  [ne](#) NE overloads the ~= operator for ltpda objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % EQ overloads the == operator for ltpda objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: EQ overloads the == operator for ltpda objects.
0005 %
0006 %           All fields are checked.
0007 %
0008 % CALL:      result = eq(obj1,obj2)
0009 %           result = eq(obj1,obj2,   exc_list)
0010 %           result = eq(obj1,obj2, 'property1', 'property2')
0011 %           result = eq(obj1,obj2, '<class>/property', '<class>/property')
0012 %
0013 % EXAMPLES:  result = eq(obj1,obj2, 'name', 'created')
0014 %           result = eq(obj1,obj2, '<class>/name')
0015 %
0016 % INPUTS:    obj1, obj2 - Input objects
0017 %           exc_list - Exception list
0018 %                   List of properties which are not checked.
0019 %
0020 % OUTPUTS:   If the two objects are considered equal, result == 1,
0021 %           otherwise, result == 0.
0022 %
0023 % M-FILE INFO: Get information about this methods by calling
0024 %           >> ao.getInfo('eq')
0025 %
0026 %           Get information about a specified set-plist by calling:
0027 %           >> ao.getInfo('eq', 'set')
0028 %
0029 % VERSION:    $Id: eq.m,v 1.8 2008/09/04 15:29:30 ingo Exp $
0030 %
0031 % HISTORY:    18-02-2008 Diepholz
0032 %           Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035
0036 function result = eq(obj1, obj2, varargin)
0037
0038 % Check if this is a call for parameters
0039 hh = {obj1, obj2, varargin{:}};
0040 if utils.helper.isinfocall(hh{:})
0041     result = getInfo(varargin{1});
0042     return
0043 end
0044
0045 import utils.const.*
0046
0047 %%%% Check class
0048 if ~strcmp(class(obj1), class(obj2))
0049     utils.helper.msg(msg.PROC1, 'NOT EQUAL: The objects are not from the same class.
[%s] <-> [%s]', class(obj1), class(obj2));
0050     result = 0;
0051     return
0052 end
0053
0054 %%%% Check length of obj1 and obj2
0055 if numel(obj1) ~= numel(obj2)
0056     utils.helper.msg(msg.PROC1, 'NOT EQUAL: The size of the %s-object''s. [%d] <->
[%d]', class(obj1), numel(obj1), numel(obj2));
0057     result = 0;
0058     return
0059 end
0060
0061 exception_list = varargin;
0062
0063 %%%% Convert a potential existing plist into a exception
0064 if ~isempty(exception_list) && isa(exception_list{1}, 'plist')
0065     exception_list = find(exception_list{1}, 'Exceptions');
0066     if isempty(exception_list)
0067         exception_list = cell(0);
0068     elseif ~iscell(exception_list)
0069         exception_list = cellstr(exception_list);
0070     end
0071 end
0072
0073 %%%% for each element in obj1 and obj2
0074 for jj = 1:numel(obj1)
0075
0076     result = 1;
0077
0078     fields = fieldnames(obj1(jj));
0079
0080     for ii = 1:length(fields)

```



```

0081     field    = fields{ii};
0082
0083     %%%%% Creates the exception list for the current field.
0084     %%%%% For example: {'name', 'ao/name'}
0085     ck_field = {field, sprintf('%s/%s', class(obj1), field)};
0086
0087     % Special case (for the ao- and history-class):
0088     % Is the field = 'hist', 'inhists' then add 'history' to the exception list.
0089     %%%%% For example: {'history', 'ao/history'}
0090     if ismember(field, {'hist', 'inhists'})
0091         ck_field{end+1} = 'history';
0092         ck_field{end+1} = sprintf('%s/history', class(obj1));
0093     elseif strcmp(field, 'val')
0094         ck_field{end+1} = 'value';
0095         ck_field{end+1} = sprintf('%s/value', class(obj1));
0096     end
0097
0098     %%%%% Check field if it is not in the exception list
0099     if ~(any(ismember(ck_field, exception_list)))
0100
0101         if isa(obj1(jj).(field), 'sym')
0102             %%%%% The property is a sym-object %%%%%%%%%%
0103             if ~eq(obj1(jj).(field), obj2(jj).(field))
0104                 result = 0;
0105                 if numel(obj1(jj)) > 1
0106                     utils.helper.msg(msg.PROC1, 'NOT EQUAL: %s.%s (%d. object)',
class(obj1(jj)), field, jj);
0107                 else
0108                     utils.helper.msg(msg.PROC1, 'NOT EQUAL: %s.%s', class(obj1(jj)), field);
0109                 end
0110                 return
0111             end
0112
0113             elseif isobject(obj1(jj).(field))
0114                 %%%%% The property is a ltpda-object %%%%%%%%%%
0115
0116                 %%%%% Check the length of the property
0117                 if length(obj1(jj).(field)) ~= length(obj2(jj).(field))
0118                     utils.helper.msg(msg.PROC1, 'NOT EQUAL: The property [%s] of the object [%s]
have not the same size', field, class(obj1));
0119                     result = 0;
0120                     return
0121                 end
0122
0123                 %%%%% For each element of the property
0124                 for kk = 1:numel(obj1(jj).(field))
0125                     if ~eq(obj1(jj).(field)(kk), obj2(jj).(field)(kk), exception_list{:})
0126                         result = 0;
0127                         if numel(obj1(jj).(field)) > 1
0128                             utils.helper.msg(msg.PROC1, 'NOT EQUAL: %s.%s (%d. element)',
class(obj1(jj)), field, kk);
0129                         else
0130                             utils.helper.msg(msg.PROC1, 'NOT EQUAL: %s.%s', class(obj1(jj)), field);
0131                         end
0132                         return
0133                     end
0134                 end
0135
0136                 else
0137                     %%%%% The property is an elemental MATLAB datatype %%%%%%%%%%
0138                     if ~isequalwithequalnans(obj1(jj).(field), obj2(jj).(field))
0139                         result = 0;
0140                         if numel(obj1(jj)) > 1
0141                             utils.helper.msg(msg.PROC1, 'NOT EQUAL: %s.%s (%d. object)',
class(obj1(jj)), field, jj);
0142                         else
0143                             utils.helper.msg(msg.PROC1, 'NOT EQUAL: %s.%s', class(obj1(jj)), field);
0144                         end
0145                         return
0146                     end
0147                 end
0148             end
0149         end
0150     end
0151 end
0152 end
0153 end
0154
0155 %%%%%%%%%% Local Functions %%%%%%%%%%
0156 %
0157 %%%%%%%%%%
0158
0159 %%%%%%%%%%
0160 %
0161 % FUNCTION:    getInfo
0162 %

```

```
0163 % DESCRIPTION: Get Info Object
0164 %
0165 % HISTORY:      11-07-07 M Hewitson
0166 %              Creation.
0167 %
0168 %%%%%%%%%%%
0169
0170 function ii = getInfo(varargin)
0171     if nargin == 1 && strcmpi(varargin{1}, 'None')
0172         sets = {};
0173         pl    = [];
0174     else
0175         sets = {'Default'};
0176         pl    = getDefaultPlist;
0177     end
0178     % Build info object
0179     ii = minfo(mfilename, 'ltpda_obj', '', utils.const.categories.relop, '$Id: eq.m,v 1.8
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0180 end
0181
0182 %%%%%%%%%%%
0183 %
0184 % FUNCTION:      getDefaultPlist
0185 %
0186 % DESCRIPTION: Get Default Plist
0187 %
0188 % HISTORY:      11-07-07 M Hewitson
0189 %              Creation.
0190 %
0191 %%%%%%%%%%%
0192
0193 function plo = getDefaultPlist()
0194     plo = plist();
0195 end
0196
```

ne

PURPOSE ^

NE overloads the ~= operator for ltpda objects.

SYNOPSIS ^

function result = ne(obj1, obj2, varargin)

DESCRIPTION ^

```
NE overloads the ~= operator for ltpda objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: NE overloads the ~= operator for ltpda objects.

CALL:
    result = ne(obj1,obj2)
    result = ne(obj1,obj2, exc_list)
    result = ne(obj1,obj2, 'property1', 'property2')
    result = ne(obj1,obj2, 'class/property', 'class/property')
    result = ne(obj1,obj2, plist('Exceptions', 'exception-list'))

PLIST:
    key:    'Exceptions'
    value:  String or cell-array with exceptions.

EXAMPLES:
    result = ne(obj1,obj2, 'name', 'created')
    result = ne(obj1,obj2, 'ao/name')

INPUTS:
    obj1,obj2 - input objects
    exc_list  - exception list
               List of properties which are not checked.

OUTPUTS:
    If the two objects are considered equal, result == 0,
    otherwise, result == 1.

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('ne')

               Get information about a specified set-plist by calling:
    >> ao.getInfo('ne', 'set')

VERSION:      $Id: ne.m,v 1.5 2008/09/04 15:29:30 ingo Exp $

HISTORY:      29-08-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [eq](#) EQ overloads the == operator for ltpda objects.

This function is called by:

 [ltpda_obj](#) LTPDA_OBJ is the abstract ltpda base class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % NE overloads the ~= operator for ltpda objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: NE overloads the ~= operator for ltpda objects.
0005 %
0006 % CALL:      result = ne(obj1,obj2)
0007 %            result = ne(obj1,obj2,  exc_list)
0008 %            result = ne(obj1,obj2,  'property1', 'property2')
0009 %            result = ne(obj1,obj2,  'class/property', 'class/property')
0010 %            result = ne(obj1,obj2,  plist('Exceptions', 'exception-list'))
0011 %
0012 % PLIST:      key:    'Exceptions'
0013 %            value:   String or cell-array with exceptions.
0014 %
0015 % EXAMPLES:   result = ne(obj1,obj2, 'name', 'created')
0016 %            result = ne(obj1,obj2, 'ao/name')
0017 %
0018 % INPUTS:     obj1,obj2 - input objects
0019 %            exc_list - exception list
0020 %                  List of properties which are not checked.
0021 %
0022 % OUTPUTS:    If the two objects are considered equal, result == 0,
0023 %            otherwise, result == 1.
0024 %
0025 % M-FILE INFO: Get information about this methods by calling
0026 %            >> ao.getInfo('ne')
0027 %
0028 %            Get information about a specified set-plist by calling:
0029 %            >> ao.getInfo('ne', 'set')
0030 %
0031 % VERSION:    $Id: ne.m,v 1.5 2008/09/04 15:29:30 ingo Exp $
0032 %
0033 % HISTORY:    29-08-2007 M Hewitson
0034 %            Creation
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037
0038 function result = ne(obj1, obj2, varargin)
0039
0040 % Check if this is a call for parameters
0041 hh = {obj1, obj2, varargin{:}};
0042 if utils.helper.isinfocall(hh{:})
0043     result = getInfo(varargin{1});
0044     return
0045 end
0046
0047 if eq(obj1, obj2, varargin{:})
0048     result = 0;
0049 else
0050     result = 1;
0051 end
0052 end
0053
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %                                Local Functions                                %
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059 %
0060 % FUNCTION:    getInfo
0061 %
0062 % DESCRIPTION: Get Info Object
0063 %
0064 % HISTORY:    11-07-07 M Hewitson
0065 %            Creation.
0066 %
0067 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0068
0069 function ii = getInfo(varargin)
0070     if nargin == 1 && strcmpi(varargin{1}, 'None')
0071         sets = {};
0072         pl = [];
0073     else
0074         sets = {'Default'};
0075         pl = getDefaultPlist;
0076     end
0077     % Build info object
0078     ii = minfo(mfilename, 'ltpda_obj', '', utils.const.categories.relop, '$Id: ne.m,v 1.5
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0079 end
0080
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0082 %  
0083 % FUNCTION:      getDefaultPlist  
0084 %  
0085 % DESCRIPTION: Get Default Plist  
0086 %  
0087 % HISTORY:       11-07-07 M Hewitson  
0088 %                Creation.  
0089 %  
0090 %%%%%%%%%%%%%%%  
0091  
0092 function plo = getDefaultPlist()  
0093     plo = plist();  
0094 end  
0095
```

cohere

PURPOSE ^

COHERE makes coherence estimates of the time-series objects

SYNOPSIS ^

function varargout = cohere(varargin)

DESCRIPTION ^

```
COHERE makes coherence estimates of the time-series objects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COHERE makes coherence estimates of the time-series objects
              in the input analysis objects. Coherences are computed using
              code based on MATLAB's mscohere (>> help mscohere).

CALL:        b = cohere(a1,a2,a3,...,pl)

INPUTS:      b      - output analysis objects
              aN     - input analysis objects (at least two)
              pl     - input parameter list

              The function makes coherence estimates between all
              input AOs. Therefore, if the input argument list contains N
              analysis objects, the output, b, will contain NXN coherence estimates.

              If the last input argument is a parameter list (plist) it is used.
              The following parameters are recognised.

PARAMETERS:  Win      - a specwin window object [default: Kaiser -200dB psll]
              Olap    - segment percent overlap [default: taken from window function]
              Nfft    - number of samples in each fft [default: half length of input data]
              Order   - order of detrending.
                        -1 - no detrending
                        0 - subtract mean [default]
                        1 - subtract linear fit
                        N - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('cohere')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('cohere', 'None')

VERSION:     $Id: cohere.m,v 1.10 2008/09/05 11:05:28 ingo Exp $

HISTORY:     07-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```

0001 % COHERE makes coherence estimates of the time-series objects
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COHERE makes coherence estimates of the time-series objects
0005 %               in the input analysis objects. Coherences are computed using
0006 %               code based on MATLAB's mscohere (>> help mscohere).
0007 %
0008 % CALL:         b = cohere(a1,a2,a3,...,pl)
0009 %
0010 % INPUTS:       b      - output analysis objects
0011 %               aN     - input analysis objects (at least two)
0012 %               pl     - input parameter list
0013 %
0014 %               The function makes coherence estimates between all
0015 %               input AOs. Therefore, if the input argument list contains N
0016 %               analysis objects, the output, b, will contain NXN coherence estimates.
0017 %
0018 %               If the last input argument is a parameter list (plist) it is used.
0019 %               The following parameters are recognised.
0020 %
0021 % PARAMETERS:   Win    - a specwin window object [default: Kaiser -200dB psll]
0022 %               Olap   - segment percent overlap [default: taken from window function]
0023 %               Nfft   - number of samples in each fft [default: half length of input
0024 %               data]
0025 %               Order  - order of detrending.
0026 %                       -1 - no detrending
0027 %                       0 - subtract mean [default]
0028 %                       1 - subtract linear fit
0029 %                       N - subtract fit of polynomial, order N
0030 % M-FILE INFO:  Get information about this methods by calling
0031 %               >> ao.getInfo('cohere')
0032 %
0033 %               Get information about a specified set-plist by calling:
0034 %               >> ao.getInfo('cohere', 'None')
0035 %
0036 % VERSION:      $Id: cohere.m,v 1.10 2008/09/05 11:05:28 ingo Exp $
0037 %
0038 % HISTORY:      07-02-2007 M Hewitson
0039 %               Creation
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %
0043 function varargout = cohere(varargin)
0044 %
0045 %% Check if this is a call for parameters
0046 if utils.helper.isinfocall(varargin{:})
0047     varargout{1} = getInfo(varargin{3});
0048     return
0049 end
0050 %
0051 import utils.const.*
0052 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0053 %
0054 % Collect input variable names
0055 in_names = cell(size(varargin));
0056 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0057 %
0058 % Collect all AOs and plists
0059 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0060 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0061 %
0062 % combine plists
0063 pl = combine(pl, getDefaultPlist());
0064 %
0065 if nargout == 0
0066     error('### cohere cannot be used as a modifier. Please give an output variable.');
```

```
0084 ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: cohere.m,v 1.10
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0085 end
0086
0087 %-----
0088 % Get Default Plist
0089 %-----
0090 function pl = getDefaultPlist()
0091     pl = plist(...
0092         'Nfft', -1, ...
0093         'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
0094         'Olap', -1, ...
0095         'Order', 0);
0096 end
0097 % END
0098
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

compute

PURPOSE ^

COMPUTE performs the given operations on the input AOs.

SYNOPSIS ^

function varargout = compute(varargin)

DESCRIPTION ^

```
COMPUTE performs the given operations on the input AOs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COMPUTE performs the given operations on the input AOs.

This is a transparent wrapper for the user selected operations and as such
doesn't add history.

CALL:      b = compute(a)
           b = compute(a, pl)
           b = compute(a, 'a(1) + a(2)./a(3)')
           b = compute(a, {'a(1) + a(2)./a(3)', 'log10(a(1))'})

PARAMETERS: 'Operations' - a string array describing the operations to
                    be performed. The input AOs are collected
                    together into a vector called 'a' and as
                    such should be so represented in your
                    operation description.

If no operation is input, then the output is just a copy of the inputs.

EXAMPLES: 1) Add the two AOs, x and y, together
           >> b = compute(x,y, plist('Operations', 'a(1) + a(2)'))
           or
           >> b = compute(x,y, 'a(1) + a(2)')

           2) Perform two operations such that the output, b, contains two AOs
           >> b = compute([x y], z, plist('Operations', {'2.*a(3)./a(1)', 'a(2)-a(1)'}))

M-FILE INFO: Get information about this methods by calling
           >> ao.getInfo('compute')

           Get information about a specified set-plist by calling:
           >> ao.getInfo('compute', 'None')

VERSION:      $Id: compute.m,v 1.11 2008/09/05 11:05:28 ingo Exp $

HISTORY:      12-03-07 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

SUBFUNCTIONS ^

 function ii = getInfo(varargin)
 function pl = getDefaultPlist()

SOURCE CODE [^]

```

0001 % COMPUTE performs the given operations on the input AOs.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COMPUTE performs the given operations on the input AOs.
0005 %
0006 % This is a transparent wrapper for the user selected operations and as such
0007 % doesn't add history.
0008 %
0009 % CALL:          b = compute(a)
0010 %               b = compute(a, pl)
0011 %               b = compute(a, 'a(1) + a(2)./a(3)')
0012 %               b = compute(a, {'a(1) + a(2)./a(3)', 'log10(a(1))'})
0013 %
0014 % PARAMETERS:   'Operations' - a string array describing the operations to
0015 %               be performed. The input AOs are collected
0016 %               together into a vector called 'a' and as
0017 %               such should be so represented in your
0018 %               operation description.
0019 %
0020 % If no operation is input, then the output is just a copy of the inputs.
0021 %
0022 % EXAMPLES: 1) Add the two AOs, x and y, together
0023 %             >> b = compute(x,y, plist('Operations', 'a(1) + a(2)'))
0024 %             or
0025 %             >> b = compute(x,y, 'a(1) + a(2)')
0026 %
0027 %             2) Perform two operations such that the output, b, contains two AOs
0028 %             >> b = compute([x y], z, plist('Operations', {'2.*a(3)./a(1)', 'a(2)-
0029 %             a(1)'}))
0030 %
0031 % M-FILE INFO:  Get information about this methods by calling
0032 %               >> ao.getInfo('compute')
0033 %
0034 %               Get information about a specified set-plist by calling:
0035 %               >> ao.getInfo('compute', 'None')
0036 %
0037 % VERSION:      $Id: compute.m,v 1.11 2008/09/05 11:05:28 ingo Exp $
0038 %
0039 % HISTORY:      12-03-07 M Hewitson
0040 %               Creation
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %
0043 function varargout = compute(varargin)
0044 %
0045 % Check if this is a call for parameters
0046 if utils.helper.isinfocall(varargin{:})
0047     varargout{1} = getInfo(varargin{3});
0048     return
0049 end
0050 %
0051 import utils.const.*
0052 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0053 %
0054 % Collect input variable names
0055 in_names = cell(size(varargin));
0056 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0057 %
0058 % Collect all AOs and plists
0059 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0060 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0061 %
0062 if nargout == 0
0063     error('### cohore cannot be used as a modifier. Please give an output variable.');
```

```

0077     ops = find(pl, 'Operations');
0078     end
0079
0080     % Check we have a cell array
0081     if ischar(ops)
0082         ops = {ops};
0083     end
0084
0085     % Loop over operations
0086     bs(length(ops),1) = ao;
0087     for jj=1:length(ops)
0088         % evaluate this operation
0089         if ops{jj}(end) ~= ':'
0090             ops{jj}(end+1) = ':';
0091         end
0092         bs(jj) = eval(ops{jj});
0093     end
0094
0095     % Set outputs
0096     varargout{1} = bs;
0097 end
0098
0099 %-----
0100 % Get Info Object
0101 %-----
0102 function ii = getInfo(varargin)
0103     if nargin == 1 && strcmpi(varargin{1}, 'None')
0104         sets = {};
0105         pl = [];
0106     else
0107         sets = {'Default'};
0108         pl = getDefaultPlist;
0109     end
0110     % Build info object
0111     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: compute.m,v 1.11
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0112 end
0113
0114 %-----
0115 % Get Default Plist
0116 %-----
0117 function pl = getDefaultPlist()
0118     pl = plist('Operations', 'a');
0119 end
0120 % END
0121
0122

```

consolidate

PURPOSE ^

CONSOLIDATE resamples all input AOs onto the same time grid.

SYNOPSIS ^

function varargout = consolidate(varargin)

DESCRIPTION ^

```
CONSOLIDATE resamples all input AOs onto the same time grid.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

CONSOLIDATE resamples all input AOs onto the same time grid and truncates all
time-series to start at the maximum start time of the inputs and end
at the minimum stop time of the inputs.

CALL:      >> bs = consolidate(as)

INPUTS:    as  - array of at least two time-series analysis objects
            pl  - parameter list (see below)

OUTPUTS:    bs  - array of analysis objects, one for each input

PARAMETER LIST:
            'fs' - specify a new sample rate for the resample grid.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('consolidate')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('consolidate', 'None')

VERSION:    $Id: consolidate.m,v 1.14 2008/09/05 11:05:28 ingo Exp $

HISTORY:    21-05-2008 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [dropduplicates](#) DROPDUPLICATES drops all duplicate samples in time-series AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fixfs](#) FIXFS resamples the input time-series to have a fixed sample rate.
-  [fs](#) FS Get the data property 'fs'.
-  [interp](#) INTERP interpolate the values in the input AO(s) at new values.
-  [interpmisssing](#) INTERPMISSING interpolate missing samples in a time-series.
-  [resample](#) RESAMPLE overloads resample function for AOs.
-  [split](#) SPLIT split an analysis object into the specified segments.
-  [t0](#) T0 Get the data property 't0'.

This function is called by:

SUBFUNCTIONS ^

 `function ii = getInfo(varargin)`
 `function pl_default = getDefaultPlist()`

SOURCE CODE

```

0001 % CONSOLIDATE resamples all input AOs onto the same time grid.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % CONSOLIDATE resamples all input AOs onto the same time grid and truncates all
0005 %         time-series to start at the maximum start time of the inputs and end
0006 %         at the minimum stop time of the inputs.
0007 %
0008 % CALL:      >> bs = consolidate(as)
0009 %
0010 % INPUTS:    as - array of at least two time-series analysis objects
0011 %            pl - parameter list (see below)
0012 %
0013 % OUTPUTS:   bs - array of analysis objects, one for each input
0014 %
0015 % PARAMETER LIST:
0016 %            'fs' - specify a new sample rate for the resample grid.
0017 %
0018 %
0019 % M-FILE INFO: Get information about this methods by calling
0020 %              >> ao.getInfo('consolidate')
0021 %
0022 %              Get information about a specified set-plist by calling:
0023 %              >> ao.getInfo('consolidate', 'None')
0024 %
0025 % VERSION:    $Id: consolidate.m,v 1.14 2008/09/05 11:05:28 ingo Exp $
0026 %
0027 % HISTORY:    21-05-2008 M Hewitson
0028 %              Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031 %
0032 %            't' - specify a new time vector to resample on to. This
0033 %                  will be truncated to fit within the maximum start
0034 %                  time and minimum stop time of the inputs.
0035 %            or
0036 %
0037 function varargout = consolidate(varargin)
0038 %
0039 % Check if this is a call for parameters
0040 if utils.helper.isinfocall(varargin{:})
0041     varargout{1} = getInfo(varargin{3});
0042     return
0043 end
0044 %
0045 import utils.const.*
0046 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0047 %
0048 % Collect input variable names
0049 in_names = cell(size(varargin));
0050 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0051 %
0052 % Collect all AOs and plists
0053 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0054 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0055 %
0056 if numel(as) < 2
0057     error('### Consolidate requires at least two time-series AOs to work.');
```

```

0078
0079 %----- Drop all repeated samples
0080 utils.helper.msg(msg.PROC1, 'drop duplicates');
0081 for j=1:na
0082     utils.helper.msg(msg.PROC2, 'processing %s', bs(j).name);
0083     dropduplicates(bs(j));
0084 end
0085
0086 %----- Interpolate all missing samples
0087 utils.helper.msg(msg.PROC1, 'interpolate missing samples');
0088 for j=1:na
0089     utils.helper.msg(msg.PROC2, 'processing %s', bs(j).name);
0090     interpmissing(bs(j));
0091 end
0092
0093 %----- Fix uneven sampling
0094 utils.helper.msg(msg.PROC1, 'fixing uneven sample rates');
0095 for j=1:na
0096     utils.helper.msg(msg.PROC2, 'processing %s', bs(j).name);
0097     fixfs(bs(j));
0098 end
0099 %----- Resample all vectors to same fs
0100 utils.helper.msg(msg.PROC1, 'resample to same fs');
0101 % If fs is specified, use it. Otherwise, use max of all
0102 % input AOs.
0103 fs = find(pl, 'fs');
0104 if isempty(fs)
0105     % compute max fs
0106     fs = 0;
0107     for j=1:na
0108         if bs(j).data.fs > fs
0109             fs = bs(j).data.fs;
0110         end
0111     end
0112 end
0113 utils.helper.msg(msg.PROC2, 'resampling all time-series to an fs of %f', fs);
0114
0115 for j=1:na
0116     % Check the resampling factor
0117     [P,Q] = utils.math.intfact(fs,bs(j).data.fs);
0118     if P > 100 || Q > 100
0119         utils.helper.msg(msg.PROC2, 'resampling factor too high [%g/%g]. Trying
interpolation', P, Q);
0120         N = length(bs(j).data.getX);
0121         t0 = bs(j).data.getX(1);
0122         t = linspace(t0, t0+(P*N/Q-1)/fs, P*N/Q);
0123         interp(bs(j), plist('vertices', t));
0124     else
0125         resample(bs(j), plist('fsout', fs));
0126     end
0127 end
0128
0129 %----- Resample all vectors on to the same grid
0130 utils.helper.msg(msg.PROC1, 'resample to same grid');
0131 % compute new time grid
0132 for j=1:na
0133     N = length(bs(j).data.getX);
0134     t = linspace(0, (N-1)/fs, N);
0135     interp(bs(j), plist('vertices', t));
0136 end
0137
0138 %----- Time properties of AOs
0139 % Find max start time
0140 start = 0;
0141 for j=1:na
0142     dstart = bs(j).data.t0.utc_epoch_milli/1000 + bs(j).data.getX(1);
0143     if dstart > start
0144         start = dstart;
0145     end
0146 end
0147
0148 % Find min stop time
0149 stop = 1e20;
0150 for j=1:na
0151     dstop = floor(bs(j).data.t0.utc_epoch_milli/1000 + bs(j).data.getX(end));
0152     if dstop < stop
0153         stop = dstop;
0154     end
0155 end
0156
0157 %----- Truncate all vectors
0158
0159 utils.helper.msg(msg.PROC1, 'truncate all vectors');
0160 utils.helper.msg(msg.PROC2, 'truncating vectors on interval [%g,%g]', start, stop);
0161
0162

```

```

0163 % split each ao
0164 bs = split(bs, plist('split_type', 'times', 'times', [start stop]));
0165
0166 nsecs = [];
0167 for j=1:na
0168     if isempty(nsecs)
0169         nsecs = bs(j).data.nsecs;
0170     end
0171     if nsecs ~= bs(j).data.nsecs
0172         error('### Something went wrong with the truncation. Vectors don't span the same
time period.');
```

```

0173     end
0174 end
0175
0176 %----- Set history on output AOs
0177
0178 for j=1:na
0179     bs(j).addHistory(getInfo, pl, ao_invars(j), inhists(j));
0180     bs(j).setName(sprintf('%s(%s)', mfilename, ao_invars{j})), 'internal');
0181 end
0182
0183 if nargout > 0
0184     varargout{1} = bs;
0185 end
0186 end
0187
0188 %-----
0189 % Get Info Object
0190 %-----
0191 function ii = getInfo(varargin)
0192     if nargin == 1 && strcmpi(varargin{1}, 'None')
0193         sets = {};
0194         pl = [];
0195     else
0196         sets = {'Default'};
0197         pl = getDefaultPlist;
0198     end
0199     % Build info object
0200     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: consolidate.m,v
1.14 2008/09/05 11:05:28 ingo Exp $', sets, pl);
0201 end
0202
0203 %-----
0204 % Get Default Plist
0205 %-----
0206 function pl_default = getDefaultPlist()
0207     pl_default = plist('fs', []);
0208 end
0209

```


conv

PURPOSE ^

CONV vector convolution.

SYNOPSIS ^

function varargout = conv(varargin)

DESCRIPTION ^

```
CONV vector convolution.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CONV vector convolution.

CALL:      >> c = conv(a,b)

INPUTS:    pl   - a parameter list
            a,b  - input analysis object

OUTPUTS:    c    - output analysis object containing the filtered data.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('conv')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('conv', 'None')

VERSION:    $Id: conv.m,v 1.1 2008/08/21 09:01:03 hewitson Exp $

HISTORY:    11-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [conv](#) CONV vector convolution.

This function is called by:

-  [conv](#) CONV vector convolution.
-  [conv_noise](#)gen CONV_NOISEGEN calls the matlab function conv.m to convolute poles and zeros from a given pzmodel

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl_default = getDefaultPlist()

SOURCE CODE ^

```
0001 % CONV vector convolution.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



```

0003 %
0004 % DESCRIPTION: CONV vector convolution.
0005 %
0006 % CALL:      >> c = conv(a,b)
0007 %
0008 % INPUTS:    pl   - a parameter list
0009 %            a,b  - input analysis object
0010 %
0011 % OUTPUTS:
0012 %            c    - output analysis object containing the filtered data.
0013 %
0014 %
0015 % M-FILE INFO: Get information about this methods by calling
0016 %              >> ao.getInfo('conv')
0017 %
0018 %              Get information about a specified set-plist by calling:
0019 %              >> ao.getInfo('conv', 'None')
0020 %
0021 % VERSION:   $Id: conv.m,v 1.1 2008/08/21 09:01:03 hewitson Exp $
0022 %
0023 % HISTORY:   11-02-07 M Hewitson
0024 %            Creation
0025 %
0026 %%%%%%%%%%%
0027
0028 function varargout = conv(varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 import utils.const.*
0037 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0038
0039 % Collect input variable names
0040 in_names = cell(size(varargin));
0041 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0042
0043 % Collect all AOs and plists
0044 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0045
0046 if nargin == 0
0047     error('### coerce cannot be used as a modifier. Please give an output variable.');
```

```

0048 end
0049
0050 if numel(as) < 2
0051     error('### conv requires at least two input AOs to work.');
```

```

0052 end
0053
0054 % Convolute the data
0055 name = ao_invars{1};
0056 res = as(1).data.y;
0057 for j=2:numel(as)
0058     res = conv(res, as(j).data.y);
0059     name = sprintf('%s,%s', name, ao_invars{j});
0060 end
0061
0062 % Make new AO
0063 bs = ao(res);
0064 % add history
0065 bs.addHistory(getInfo, getDefaultPlist, ao_invars, [as(:).hist]);
0066 % name for this object
0067 bs.setName(sprintf('conv(%s)', name), 'internal');
```

```

0068
0069 % Set outputs
0070 if nargin == 1
0071     varargout{1} = bs;
0072 else
0073     error('### wrong number of output arguments.');
```

```

0074 end
0075 end
0076
0077 %-----
0078 % Get Info Object
0079 %-----
0080 function ii = getInfo(varargin)
0081 if nargin == 1 && strcmpi(varargin{1}, 'None')
0082     sets = {};
0083     pls  = [];
0084 else
0085     sets = {'Default'};
0086     pls  = getDefaultPlist;
0087 end
0088 % Build info object

```

```
0089   ii = minfo(mfilename, 'ao', '', 'Signal Processing', '$Id: conv.m,v 1.1 2008/08/21
09:01:03 hewitson Exp $', sets, pls);
0090 end
0091
0092 %-----
0093 % Get Default Plist
0094 %-----
0095 function pl_default = getDefaultPlist()
0096     pl_default = plist();
0097 end
0098
```

cpsd

PURPOSE ^

CPSD makes cross-spectral density estimates of the time-series objects.

SYNOPSIS ^

function varargout = cpsd(varargin)

DESCRIPTION ^

CPSD makes cross-spectral density estimates of the time-series objects.
%%%

DESCRIPTION: CPSD makes cross-spectral density estimates of the time-series objects in the input analysis objects. CPSDs are computed using a modified version of MATLAB's cpsd (>> help cpsd).

CALL: b = cpsd(a1,a2,a3,...,pl)

INPUTS: b - output analysis objects
aN - input analysis objects (at least two)
pl - input parameter list

The function makes CPSD estimates between all input AOs. Therefore, if the input argument list contains N analysis objects, the output a will contain NxN CPSD estimates. The diagonal elements will be S_ai_ai and will be equivalent to the output of ltpda_pwelch(ai)

If the last input argument is a parameter list (plist) it is used. The following parameters are recognised.

PARAMETERS: 'Win' - a specwin window object [default: Kaiser -200dB psll]
'Olap' - segment percent overlap [default: taken from window function]
'Nfft' - number of samples in each fft [default: length of input data]
'Order' - order of segment detrending:
-1 - no detrending
0 - subtract mean [default]
1 - subtract linear fit
N - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('cpsd')

Get information about a specified set-plist by calling:
>> ao.getInfo('cpsd', 'None')

VERSION: \$Id: cpsd.m,v 1.9 2008/09/05 11:05:28 ingo Exp \$

HISTORY: 07-02-2007 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```

0001 % CPSD makes cross-spectral density estimates of the time-series objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CPSD makes cross-spectral density estimates of the
0005 %               time-series objects in the input analysis objects. CPSDs are computed
0006 %               using a modified version of MATLAB's cpsd (>> help cpsd).
0007 %
0008 % CALL:         b = cpsd(a1,a2,a3,...,pl)
0009 %
0010 % INPUTS:       b      - output analysis objects
0011 %               aN     - input analysis objects (at least two)
0012 %               pl     - input parameter list
0013 %
0014 %               The function makes CPSD estimates between all input AOs.
0015 %               Therefore, if the input argument list contains N analysis objects, the
0016 %               output a will contain NxN CPSD estimates.
0017 %               The diagonal elements will be S_ai_ai and will be equivalent to
0018 %               the output of ltpda_pwelch(ai)
0019 %
0020 %               If the last input argument is a parameter list (plist) it is used. The
0021 %               following parameters are recognised.
0022 %
0023 % PARAMETERS:   'Win'    - a specwin window object [default: Kaiser -200dB psll]
0024 %               'Olap'   - segment percent overlap [default: taken from window function]
0025 %               'Nfft'   - number of samples in each fft [default: length of input data]
0026 %               'Order'  - order of segment detrending:
0027 %                       -1 - no detrending
0028 %                       0 - subtract mean [default]
0029 %                       1 - subtract linear fit
0030 %                       N - subtract fit of polynomial, order N
0031 %
0032 % M-FILE INFO:  Get information about this methods by calling
0033 %               >> ao.getInfo('cpsd')
0034 %
0035 %               Get information about a specified set-plist by calling:
0036 %               >> ao.getInfo('cpsd', 'None')
0037 %
0038 % VERSION:      $Id: cpsd.m,v 1.9 2008/09/05 11:05:28 ingo Exp $
0039 %
0040 % HISTORY:      07-02-2007 M Hewitson
0041 %               Creation
0042 %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044
0045 function varargout = cpsd(varargin)
0046
0047 % Check if this is a call for parameters
0048 if utils.helper.isinfocall(varargin{:})
0049     varargout{1} = getInfo(varargin{3});
0050     return
0051 end
0052
0053 import utils.const.*
0054 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0055
0056 % Collect input variable names
0057 in_names = cell(size(varargin));
0058 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0059
0060 % Collect all AOs and plists
0061 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0062 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0063
0064 if nargout == 0
0065     error('### cohere cannot be used as a modifier. Please give an output variable.');
```

```
0083     pl     = getDefaultPlist;
0084     end
0085     % Build info object
0086     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: cpsd.m,v 1.9
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0087 end
0088
0089 %-----
0090 % Get Default Plist
0091 %-----
0092 function pl = getDefaultPlist()
0093     pl = plist(...
0094         'Nfft', -1, ...
0095         'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
0096         'Olap', -1, ...
0097         'Order', 0);
0098 end
0099
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

delay

PURPOSE ^

DELAY delays a time-series using various methods.

SYNOPSIS ^

function varargout = delay(varargin)

DESCRIPTION ^

```
DELAY delays a time-series using various methods.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DELAY delays a time-series using various methods.

CALL:      b = delay(a, pl)

PARAMETERS: N    - delay the time-series by N samples. [default: 0]
               Choose a 'method' for how the end of the time-series
               is handled:
               'zero'   - zero pad the time-series

EXAMPLES:    1) Shift by 10 samples and zero pad the end of the time-series
               >> b = delay(a, plist('N', 10, 'method', 'zero'));

M-FILE INFO: Get information about this methods by calling
               >> ao.getInfo('delay')

               Get information about a specified set-plist by calling:
               >> ao.getInfo('delay', 'None')

VERSION:     $Id: delay.m,v 1.12 2008/09/05 11:05:29 ingo Exp $

HISTORY:     08-04-08 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DELAY delays a time-series using various methods.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DELAY delays a time-series using various methods.
0005 %
0006 % CALL:      b = delay(a, pl)
```

```

0007 %
0008 %
0009 % PARAMETERS:  N    - delay the time-series by N samples. [default: 0]
0010 %                  Choose a 'method' for how the end of the time-series
0011 %                  is handled:
0012 %                  'zero'    - zero pad the time-series
0013 %
0014 % EXAMPLES:      1) Shift by 10 samples and zero pad the end of the time-series
0015 %                  >> b = delay(a, plist('N', 10, 'method', 'zero'));
0016 %
0017 % M-FILE INFO:   Get information about this methods by calling
0018 %                  >> ao.getInfo('delay')
0019 %
0020 %                  Get information about a specified set-plist by calling:
0021 %                  >> ao.getInfo('delay', 'None')
0022 %
0023 % VERSION:       $Id: delay.m,v 1.12 2008/09/05 11:05:29 ingo Exp $
0024 %
0025 % HISTORY:       08-04-08 M Hewitson
0026 %                  Creation
0027 %
0028 %%%%%%%%%%%
0029 %
0030 % TODO
0031 %                  'extrap' - extrapolate the last N samples
0032 %
0033 function varargout = delay(varargin)
0034 %
0035 % Check if this is a call for parameters
0036 if utils.helper.isinfocall(varargin{:})
0037     varargout{1} = getInfo(varargin{3});
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all AOs
0049 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0051
0052 % Decide on a deep copy or a modify
0053 bs = copy(as, nargout);
0054
0055 % Combine plists
0056 pl = combine(pl, getDefaultPlist);
0057
0058 %----- Get parameters
0059
0060 % 1: Sample shift
0061 N = find(pl, 'N');
0062 method = find(pl, 'method');
0063
0064 % Loop over AOs
0065 for j=1:numel(bs)
0066     if ~isa(bs(j).data, 'tsdata')
0067         warning('!!! Skipping object %s - it contains no tsdata.', ao_invars{j});
0068     else
0069         % Which method to use
0070         switch method
0071             case 'zero'
0072                 s = size(bs(j).data.y);
0073                 if s(1) > s(2)
0074                     bs(j).setY([zeros(N,1); bs(j).data.y(1:end-N)], 'internal');
0075                 else
0076                     bs(j).setY([zeros(1,N) bs(j).data.y(1:end-N)], 'internal');
0077                 end
0078             otherwise
0079                 error('### Unknown method for dealing with end of time-series.');

```

```

0093
0094 %-----
0095 % Get Info Object
0096 %-----
0097 function ii = getInfo(varargin)
0098     if nargin == 1 && strcmpi(varargin{1}, 'None')
0099         sets = {};
0100         pl = [];
0101     else
0102         sets = {'Default'};
0103         pl = getDefaultPlist;
0104     end
0105     % Build info object
0106     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: delay.m,v 1.12
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0107 end
0108
0109 %-----
0110 % Get Default Plist
0111 %-----
0112 function pl = getDefaultPlist()
0113     pl = plist('N', 0, ...
0114         'method', 'zero');
0115 end
0116
0117

```


detrend

PURPOSE ^

DETREND detrends the input analysis object using a polynomial of degree N.

SYNOPSIS ^

function varargout = detrend(varargin)

DESCRIPTION ^

```
DETREND detrends the input analysis object using a polynomial of degree N.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DETREND detrends the input analysis object using a
             polynomial of degree N.

CALL:       b = detrend(a1,a2,a3,..., pl)

INPUTS:     aN  - a list of analysis objects
             pl  - a parameter list

OUTPUTS:    b - array of analysis objects

             If the last input argument is a parameter list (plist) it is used.
             The following parameters are recognised.

PARAMETERS:  'N' - degree of polynomial [default: 0]

M-FILE INFO: Get information about this methods by calling
             >> ao.getInfo('detrend')

             Get information about a specified set-plist by calling:
             >> ao.getInfo('detrend', 'None')

VERSION:    $Id: detrend.m,v 1.10 2008/09/05 11:05:29 ingo Exp $

HISTORY:    01-03-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [polyfit](#) POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [mlpsd_m](#) MLPSD_M m-file only version of the LPSD algorithm

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)
-  [function y = polydetrend\(varargin\)](#)

SOURCE CODE ^

```

0001 % DETREND detrends the input analysis object using a polynomial of degree N.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DETREND detrends the input analysis object using a
0005 %               polynomial of degree N.
0006 %
0007 % CALL:         b = detrend(a1,a2,a3,..., pl)
0008 %
0009 % INPUTS:       aN  - a list of analysis objects
0010 %               pl  - a parameter list
0011 %
0012 % OUTPUTS:      b - array of analysis objects
0013 %
0014 %               If the last input argument is a parameter list (plist) it is used.
0015 %               The following parameters are recognised.
0016 %
0017 % PARAMETERS:   'N' - degree of polynomial [default: 0]
0018 %
0019 % M-FILE INFO:  Get information about this methods by calling
0020 %               >> ao.getInfo('detrend')
0021 %
0022 %               Get information about a specified set-plist by calling:
0023 %               >> ao.getInfo('detrend', 'None')
0024 %
0025 % VERSION:      $Id: detrend.m,v 1.10 2008/09/05 11:05:29 ingo Exp $
0026 %
0027 % HISTORY:      01-03-2007 M Hewitson
0028 %               Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 function varargout = detrend(varargin)
0033
0034 % Check if this is a call for parameters
0035 if utils.helper.isinfocall(varargin{:})
0036     varargout{1} = getInfo(varargin{3});
0037     return
0038 end
0039
0040 import utils.const.*
0041 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0042
0043 % Collect input variable names
0044 in_names = cell(size(varargin));
0045 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0046
0047 % Collect all AOs
0048 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0049 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0050
0051 % Decide on a deep copy or a modify
0052 bs = copy(as, nargout);
0053
0054 % Combine plists
0055 pl = combine(pl, getDefaultPlist);
0056
0057 % Unpack parameter list
0058 N = find(pl, 'N');
0059
0060 % Loop over analysis objects
0061 for jj=1: numel(bs)
0062     % check data
0063     if ~isa(bs(jj).data, 'tsdata')
0064         warning('!!! This method only works with time-series at the moment. Skipping %s',
0065             ao_invars{jj});
0066     else
0067         % detrend with polynomial
0068         try
0069             if find(pl, 'M-FILE ONLY')
0070                 bs(jj).data.setY(polydetrend(bs(jj).data.y.', N));
0071             else
0072                 bs(jj).data.setY(ltpda_polyreg(bs(jj).data.y, N).');
0073             end
0074         catch
0075             warning('!!! failed to execture ltpda_polyreg.mex. Using m-file call.');
```

```

0081     end
0082 end
0083
0084 if nargin > 0
0085     varargout{1} = bs;
0086 end
0087 end
0088
0089 %-----
0090 % Get Info Object
0091 %-----
0092 function ii = getInfo(varargin)
0093
0094     if nargin == 1 && strcmpi(varargin{1}, 'None')
0095         sets = {};
0096         pl = [];
0097     else
0098         sets = {'Default'};
0099         pl = getDefaultPlist;
0100     end
0101     % Build info object
0102     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: detrend.m,v 1.10
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0103 end
0104
0105 %-----
0106 % Get Default Plist
0107 %-----
0108 function pl = getDefaultPlist()
0109     pl = plist('N', 0);
0110 end
0111
0112 function y = polydetrend(varargin)
0113     % POLYDETREND detrends the input data vector with a polynomial.
0114     %
0115     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116     %
0117     % DESCRIPTION: POLYDETREND detrends the input data vector with a polynomial.
0118     %               This function is meant to be used by other ltpda_ functions.
0119     %               To work with Analysis Objects, use ltpda_polydetrend which calls
0120     %               this function.
0121     %
0122     % CALL:         y = polydetrend(x,order);
0123     %
0124     % INPUTS:       x      - vector of x values
0125     %               order - order of polynomial to fit and subtract
0126     %
0127     % OUTPUTS:      y      - detrended data
0128     %
0129     % VERSION:      $Id: detrend.m,v 1.10 2008/09/05 11:05:29 ingo Exp $
0130     %
0131     % HISTORY:      30-05-07 M Hewitson
0132     %               Creation
0133     %
0134     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0135
0136     if nargin == 2
0137         x = varargin{1};
0138         order = varargin{2};
0139         t = [1:length(x)]';
0140     elseif nargin == 3
0141         t = varargin{1};
0142         x = varargin{2};
0143         order = varargin{3};
0144     else
0145         error('### incorrect inputs. ');
0146     end
0147
0148     % fit polynomial
0149     p = polyfit(t, x, order);
0150
0151     % make polynomial series
0152     py = polyval(p, t);
0153
0154     % detrend
0155     y = x - py;
0156 end
0157 % END
0158

```

dft

PURPOSE ^

DFT computes the DFT of the input time-series at the requested frequencies.

SYNOPSIS ^

function varargout = dft(varargin)

DESCRIPTION ^

```
DFT computes the DFT of the input time-series at the requested frequencies.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DFT computes the DFT of the input time-series at the requested
frequencies.

CALL:      b = dft(a, pl)

PARAMETERS: 'f' - vector of frequencies at which to compute the DFT.
              [default: linspace(1/ndata,fs/2,ndata/2+1)]

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('dft')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('dft', 'None')

VERSION:    $Id: dft.m,v 1.13 2008/09/05 14:14:11 hewitson Exp $

HISTORY:    25-04-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [dft](#) DFT computes the DFT of the input time-series at the requested frequencies.
-  [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [yunits](#) YUNITS Get the data property 'yunits'.

This function is called by:

-  [dft](#) DFT computes the DFT of the input time-series at the requested frequencies.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```

0001 % DFT computes the DFT of the input time-series at the requested frequencies.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DFT computes the DFT of the input time-series at the requested
0005 %               frequencies.
0006 %
0007 % CALL:         b = dft(a, pl)
0008 %
0009 % PARAMETERS:
0010 %               'f' - vector of frequencies at which to compute the DFT.
0011 %                   [default: linspace(1/ndata,fs/2,ndata/2+1)]
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> ao.getInfo('dft')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> ao.getInfo('dft', 'None')
0018 %
0019 % VERSION:      $Id: dft.m,v 1.13 2008/09/05 14:14:11 hewitson Exp $
0020 %
0021 % HISTORY:      25-04-08 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = dft(varargin)
0027
0028 % Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 import utils.const.*
0035 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0036
0037 % Collect input variable names
0038 in_names = cell(size(varargin));
0039 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0040
0041 % Collect all AOs
0042 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0043 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0044
0045 % Make copies or handles to inputs
0046 bs = copy(as, nargsout);
0047
0048 % Combine plists
0049 pl = combine(pl, getDefaultPlist);
0050
0051 % Loop over input AOs
0052 for j=1:length(bs)
0053     if ~isa(bs(j).data, 'tsdata')
0054         warning('!!! The DFT can only be computed on input time-series. Skipping AO %s',
ao_invars{j});
0055     else
0056
0057         % Extract necessary parameters
0058         f = find(pl, 'f');
0059
0060         % Compute f if necessary
0061         if f == -1
0062             f = linspace(0, bs(j).data.fs/2, length(bs(j).data.getY)/2+1);
0063         end
0064
0065         % Compute DFT
0066         fs = bs(j).data.fs;
0067         N = length(bs(j).data.getY);
0068         J = -2*pi*1i.*[0:N-1]/fs;
0069         dft = zeros(size(f));
0070         for kk = 1:length(f)
0071             dft(kk) = exp(f(kk)*J)*bs(j).data.getY;
0072         end
0073
0074         % Make output fsdata AO
0075         yunits = bs(j).data.yunits;
0076         % Keep the data shape of the input AO
0077         if size(bs(j).data.y,1) ~= 1
0078             f = f.';
0079             dft = dft.';
0080         end
0081         bs(j).data = fsdata(f, dft, bs(j).data.fs);
0082         bs(j).setXunits('Hz', 'internal');
0083         bs(j).setYunits(yunits./unit('Hz'), 'internal');
0084         % Set name
0085         bs(j).setName(sprintf('dft(%s)', ao_invars{j}), 'internal');

```

```

0086         % Add history
0087         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0088     end
0089 end
0090
0091 % Set outputs
0092 if nargout > 0
0093     varargout{1} = bs;
0094 end
0095 end
0096
0097 %-----
0098 % Get Info Object
0099 %-----
0100 function ii = getInfo(varargin)
0101     if nargin == 1 && strcmpi(varargin{1}, 'None')
0102         sets = {};
0103         pls = [];
0104     else
0105         sets = {'Default'};
0106         pls = getDefaultPlist;
0107     end
0108     % Build info object
0109     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: dft.m,v 1.13
2008/09/05 14:14:11 hewitson Exp $', sets, pls);
0110 end
0111
0112 %-----
0113 % Get Default Plist
0114 %-----
0115 function pl = getDefaultPlist()
0116     pl = plist('f', -1);
0117 end
0118
0119

```

diff

PURPOSE ^

DIFF differentiates the data in AO.

SYNOPSIS ^

function varargout = diff(varargin)

DESCRIPTION ^

DIFF differentiates the data in AO.
%%%

DESCRIPTION: DIFF differentiates the data in AO. The result is a data series the same length as the input series.

CALL: >> b = diff(a,pl)

INPUTS: pl - a parameter list
 a - input analysis object

OUTPUTS: b - output analysis object containing the differentiated data.

PARAMETERS: method - the method to use: [default: '3POINT']
 '2POINT' - 2 point derivative computed as
 [y(i+1)-y(i)]./[x(i+1)-x(i)].
 '3POINT' - 3 point derivative. Compute derivative
 at i as [y(i+1)-y(i-1)] / [x(i+1)-x(i-1)].
 For i==1, the output is computed as
 [y(2)-y(1)]/[x(2)-x(1)]. The last sample
 is computed as [y(N)-y(N-1)]/[x(N)-x(N-1)].
 '5POINT' - 5 point derivative. Compute derivative dx
 at i as
 [-y(i+2)+8*y(i+1)-8*y(i-1)+y(i-2)] /
 [3*(x(i+2)-x(i-2))].
 For i==1, the output is computed as
 [y(2)-y(1)]/[x(2)-x(1)]. The last sample
 is computed as [y(N)-y(N-1)]/[x(N)-x(N-1)].
 'ORDER2' - Compute derivative using a 2nd order
 method.
 'ORDER2SMOOTH' - Compute derivative using a 2nd order
 method with a parabolic fit to 5
 consecutive samples.
 'filter' - applies an IIR filter built from a
 single pole at the chosen frequency. The
 filter is applied forwards and backwards
 (filtfilt) to achieve the desired f^2
 response. This only works for time-series
 AOs. For this method, you can specify the
 pole frequency with an additional parameter
 'f0' [default: 1/Nsecs]

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('diff')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('diff', 'None')

VERSION: \$Id: diff.m,v 1.10 2008/09/05 11:05:29 ingo Exp \$

HISTORY: 03-07-08 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **abs** ABS overloads the Absolute value method for Analysis objects.
-  **copy** COPY makes a (deep) copy of the input AOs.
-  **diff** DIFF differentiates the data in AO.
-  **find** FIND particular samples that satisfy the input query and return a new AO.
-  **max** MAX computes the maximum value of the data in the AO.
-  **mean** MEAN computes the mean value of the data in the AO.
-  **sqrt** SQRT computes the square root of the data in the AO.
-  **sum** SUM computes the sum of the data in the AO.
-  **x** X Get the data property 'x'.
-  **y** Y Get the data property 'y'.

This function is called by:

-  **diff** DIFF differentiates the data in AO.
-  **dropduplicates** DROPDUPLICATES drops all duplicate samples in time-series AOs.
-  **gapfilling** GAPFILLING fills possible gaps in data.
-  **interpmissing** INTERPMISSING interpolate missing samples in a time-series.

SUBFUNCTIONS [^]

-  **function ii = getInfo(varargin)**
-  **function pl_default = getDefaultPlist()**

SOURCE CODE [^]

```

0001 % DIFF differentiates the data in AO.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DIFF differentiates the data in AO. The result is a data
0005 %               series the same length as the input series.
0006 %
0007 % CALL:        >> b = diff(a,pl)
0008 %
0009 % INPUTS:      pl   - a parameter list
0010 %              a    - input analysis object
0011 %
0012 % OUTPUTS:     b    - output analysis object containing the differentiated data.
0013 %
0014 %
0015 % PARAMETERS:  method - the method to use: [default: '3POINT']
0016 %              '2POINT' - 2 point derivative computed as
0017 %                        [y(i+1)-y(i)]./[x(i+1)-x(i)].
0018 %              '3POINT' - 3 point derivative. Compute derivative
0019 %                        at i as [y(i+1)-y(i-1)] / [x(i+1)-x(i-1)].
0020 %                        For i=1, the output is computed as
0021 %                        [y(2)-y(1)]/[x(2)-x(1)]. The last sample
0022 %                        is computed as [y(N)-y(N-1)]/[x(N)-x(N-1)].
0023 %              '5POINT' - 5 point derivative. Compute derivative dx
0024 %                        at i as
0025 %                        [-y(i+2)+8*y(i+1)-8*y(i-1)+y(i-2)] /
0026 %                        [3*(x(i+2)-x(i-2))].
0027 %                        For i=1, the output is computed as
0028 %                        [y(2)-y(1)]/[x(2)-x(1)]. The last sample
0029 %                        is computed as [y(N)-y(N-1)]/[x(N)-x(N-1)].
0030 %              'ORDER2' - Compute derivative using a 2nd order
0031 %                        method.
0032 %              'ORDER2SMOOTH' - Compute derivative using a 2nd order
0033 %                        method with a parabolic fit to 5
0034 %                        consecutive samples.
0035 %              'filter' - applies an IIR filter built from a
0036 %                        single pole at the chosen frequency. The
0037 %                        filter is applied forwards and backwards
0038 %                        (filtfilt) to achieve the desired f^2
0039 %                        response. This only works for time-series
0040 %                        AOs. For this method, you can specify the
0041 %                        pole frequency with an additional parameter
0042 %                        'f0' [default: 1/Nsecs]
0043 %
0044 % M-FILE INFO: Get information about this methods by calling

```



```

0045 %                >> ao.getInfo('diff')
0046 %
0047 %                Get information about a specified set-plist by calling:
0048 %                >> ao.getInfo('diff', 'None')
0049 %
0050 % VERSION:      $Id: diff.m,v 1.10 2008/09/05 11:05:29 ingo Exp $
0051 %
0052 % HISTORY:      03-07-08 M Hewitson
0053 %                Creation
0054 %
0055 %%%%%%%%%%%%%%%
0056 %
0057 function varargout = diff(varargin)
0058 %
0059 % Check if this is a call for parameters
0060 if utils.helper.isinfocall(varargin{:})
0061     varargout{1} = getInfo(varargin{3});
0062     return
0063 end
0064 %
0065 import utils.const.*
0066 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0067 %
0068 % Collect input variable names
0069 in_names = cell(size(varargin));
0070 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0071 %
0072 % Collect all AOs and plists
0073 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0074 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0075 %
0076 % Decide on a deep copy or a modify
0077 bs = copy(as, nargout);
0078 %
0079 % combine plists
0080 pl = combine(pl, getDefaultPlist());
0081 %
0082 % Extract method
0083 method = find(pl, 'method');
0084 %
0085 for j=1: numel(bs)
0086 %
0087 % Diff can't work for cdata objects since we need x data
0088 if isa(bs(j).data, 'cdata')
0089     error('### diff doesn''t work with cdata AOs since we need an x-data vector.');
```

```

0090 end
0091 %
0092 % Compute derivative with selected method
0093 switch method
0094     case '2POINT'
0095         x = bs(j).data.getX;
0096         dx = diff(x);
0097         y = bs(j).data.getY;
0098         dy = diff(y);
0099         z = dy./dx;
0100         bs(j).setY([z; 2*z(end)-z(end-1)], 'internal');
0101     case '3POINT'
0102         x = bs(j).data.getX;
0103         dx = diff(x);
0104         y = bs(j).data.getY;
0105         z = zeros(size(y));
0106         z(2:end-1) = (y(3:end)-y(1:end-2)) ./ (dx(2:end)+dx(1:end-1));
0107         z(1) = (y(2)-y(1)) ./ (dx(1));
0108         z(end) = 2*z(end-1)-z(end-2);
0109         bs(j).setY(z, 'internal');
0110     case '5POINT'
0111         x = bs(j).data.getX;
0112         dx = diff(x);
0113         y = bs(j).data.getY;
0114         z = zeros(size(y));
0115         z(1) = (y(2)-y(1)) ./ (dx(1));
0116         z(2) = (y(3)-y(1)) ./ (dx(2)+dx(1));
0117         z(3:end-2) = (-y(5:end) + 8.*y(4:end-1) - 8.*y(2:end-3) + y(1:end-4)) ./
0118         (3.*(x(5:end)-x(1:end-4)));
0119         z(end-1) = 2*z(end-2)-z(end-3);
0120         z(end) = 2*z(end-1)-z(end-2);
0121         bs(j).setY(z, 'internal');
0122     case 'ORDER2'
0123         x = bs(j).data.getX;
0124         dx = diff(x);
0125         y = bs(j).data.getY;
0126         z = zeros(size(y));
0127         m = length(y);
0128         % y'(x1)
0129         z(1) = (1/dx(1)+1/dx(2))*(y(2)-y(1))+...
0130         dx(1)/(dx(1)*dx(2)+dx(2)^2)*(y(1)-y(3));

```

```

0130 % y'(xm)
0131 z(m) = (1/dx(m-2)+1/dx(m-1))*(y(m)-y(m-1))+...
0132 dx(m-1)/(dx(m-1)*dx(m-2)+dx(m-2)^2)*(y(m-2)-y(m));
0133 % y'(xi) (i>1 & i<m)
0134 dx1 = repmat(dx(1:m-2),1,1);
0135 dx2 = repmat(dx(2:m-1),1,1);
0136 y1 = y(1:m-2); y2 = y(2:m-1); y3 = y(3:m);
0137 z(2:m-1) = 1./((dx1.*dx2.*(dx1+dx2)).*...
0138 (-dx2.^2.*y1+(dx2.^2-dx1.^2).*y2+dx1.^2.*y3);
0139 bs(j).setY(z, 'internal');
0140 case 'ORDER2SMOOTH'
0141 x = bs(j).data.getX;
0142 y = bs(j).data.getY;
0143 dx = diff(x);
0144 m = length(y);
0145 if max(abs(diff(dx)))>sqrt(eps(max(abs(dx))))
0146 error('### The x-step must be constant for method ''ORDER2SMOOTH'')
0147 elseif m<5
0148 error('### Length of y must be at least 5 for method ''ORDER2SMOOTH''.')
0149 end
0150 h = mean(dx);
0151 z = zeros(size(y));
0152 % y'(x1)
0153 z(1) = sum(y(1:5).*[-54; 13; 40; 27; -26])/70/h;
0154 % y'(x2)
0155 z(2) = sum(y(1:5).*[-34; 3; 20; 17; -6])/70/h;
0156 % y'(x{m-1})
0157 z(m-1) = sum(y(end-4:end).*[6; -17; -20; -3; 34])/70/h;
0158 % y'(xm)
0159 z(m) = sum(y(end-4:end).*[26; -27; -40; -13; 54])/70/h;
0160 % y'(xi) (i>2 & i<(N-1))
0161 Dc = [2 1 0 -1 -2];
0162 tmp = convn(Dc,y)/10/h;
0163 z(3:m-2) = tmp(5:m);
0164 bs(j).setY(z, 'internal');
0165 case 'filter'
0166 otherwise
0167 error('### Unknown method for computing the derivative.');
```

```

0168 end
0169
0170 % add history
0171 bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0172 % name for this object
0173 bs(j).setName(sprintf('diff(%s)', ao_invars{j}), 'internal');
```

```

0174 end
0175
0176 % Set outputs
0177 if nargout > 0
0178     varargout{1} = bs;
0179 end
0180 end
0181
0182 %-----
0183 % Get Info Object
0184 %-----
0185 function ii = getInfo(varargin)
0186
0187     if nargin == 1 && strcmpi(varargin{1}, 'None')
0188         sets = {};
0189         pl = [];
0190     else
0191         sets = {'Default'};
0192         pl = getDefaultPlist;
0193     end
0194     % Build info object
0195     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: diff.m,v 1.10
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0196 end
0197
0198 %-----
0199 % Get Default Plist
0200 %-----
0201 function pl_default = getDefaultPlist()
0202     pl_default = plist('method', '3POINT');
0203 end
0204
```

dopplercorr

PURPOSE ^

FOO description for function 'foo' in one line. Necessary for lookfor functionality.

SYNOPSIS ^

function varargout = dopplercorr(varargin)

DESCRIPTION ^

```
FOO description for function 'foo' in one line. Necessary for lookfor functionality.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: dopplercorr detailed description for the function 'dopplercorr'

CALL:      b = dopplercorr(a,pl)

INPUTS:    a - analysis object(s) (time series)
           pl - parameter list(s) - bin

OUTPUTS:   b - analysis object  (time series)

M-FILE INFO: Get information about this methods by calling
              >> class.getInfo('foo')

              Get information about a specified set-plist by calling:
              >> class.getInfo('foo', 'set')

VERSION:    $Id: dopplercorr.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    28-07-2008 A Monsky
              Creation

SEE ALSO:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [sin](#) SIN overloads the sin method for Analysis objects.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % FOO description for function 'foo' in one line. Necessary for lookfor functionality.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: dopplercorr detailed description for the function 'dopplercorr'
0005 %
```

```

0006 % CALL:      b = dopplercorr(a,pl)
0007 %
0008 % INPUTS:     a - analysis object(s) (time series)
0009 %             pl - parameter list(s) - bin
0010 %
0011 % OUTPUTS:    b - analysis object (time series)
0012 %
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> class.getInfo('foo')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> class.getInfo('foo', 'set')
0019 %
0020 % VERSION:    $Id: dopplercorr.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0021 %
0022 % HISTORY:    28-07-2008 A Monsky
0023 %             Creation
0024 %
0025 % SEE ALSO:
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = dopplercorr(varargin)
0030
0031 %%% Check if this is a call for parameters
0032 if utils.helper.isinfocall(varargin{:})
0033     varargout{1} = getInfo(varargin{3});
0034     return
0035 end
0036
0037 %%% Collect input variable names
0038 in_names = cell(size(varargin));
0039 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0040
0041 %%% Collect all AOs
0042 [as, ao_invars, rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0043 [pli, pl_invars, rest] = utils.helper.collect_objects(rest, 'plist', in_names);
0044
0045 %%% Decide on a deep copy or a modify
0046 %%% REMARK: If you create a new AO (call the constructor) then
0047 %%%         it is not necessary to copy the input-AOs !!!!!!!!!!!!!!!!!!!!!!!
0048 bs = copy(as, nargout);
0049
0050 %%% Combine plists
0051 pl = combine(pli, getDefaultPlist);
0052 k = find(pl, 'bin');
0053 %%% go through analysis objects
0054 for kk = 1:numel(bs)
0055     %%%%%%%%% some calculations %%%%%%%%%
0056     ydata = bs(kk).data.getY;
0057     leng = length(ydata);
0058     delta = zeros(leng,1);
0059     delta(1) = (ydata(2)-ydata(1))/(2*pi);
0060     delta(leng) = (ydata(leng)-ydata(leng-1))/(2*pi);
0061
0062     for i=2:(leng-1)
0063         delta(i) = (ydata(i+1)-ydata(i-1))/(4*pi);
0064     end
0065
0066     g = -1./(2*k).*delta.*sin(2.*ydata)...
0067         + 1./(4*k*k).*delta.^2.*sin(2.*ydata)...
0068         + 1./(8*k*k).*delta.^2.*sin(4.*ydata);
0069     ydata_new = ydata + g;
0070
0071     %%create new ao
0072     bs(kk).data.setY(ydata_new);
0073
0074     % g_1 = delta./(2*k).*((delta./(2*k)-
0075     1).*sin(2.*ydata)+delta./(4*k).*sin(4.*ydata));
0076
0077
0078
0079 %     bs(kk).data.setY(some_new_data);
0080 %     bs(kk).data.setFs(new_fs);
0081 %     bs(kk).data.setXunits(new_xunits);
0082 %     bd(kk).setName('my name');
0083
0084 %%% Set Name
0085 bs(kk).setName('new name', 'internal');
0086
0087 %%% Add History
0088 bs(kk).addHistory(getInfo('None'), pl, ao_invars(kk), bs(kk).hist);
0089
0090 end

```

```
0091
0092   %%% Reshape the ouput to the same size of the input
0093   varargout{1} = bs;
0094
0095 end
0096
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098 %                               Local Functions                               %
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100
0101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0102 %
0103 % FUNCTION:      getInfo
0104 %
0105 % DESCRIPTION:   Get Info Object
0106 %
0107 % HISTORY:       11-07-07 M Hewitson
0108 %                  Creation.
0109 %
0110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111
0112 function ii = getInfo(varargin)
0113     if nargin == 1 && strcmpi(varargin{1}, 'None')
0114         sets = {};
0115         pl   = [];
0116     else
0117         sets = {'Default'};
0118         pl   = getDefaultPlist;
0119     end
0120     % Build info object
0121     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: dopplercorr.m,v
1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0122 end
0123
0124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0125 %
0126 % FUNCTION:      getDefaultPlist
0127 %
0128 % DESCRIPTION:   Get Default Plist
0129 %
0130 % HISTORY:       11-07-07 M Hewitson
0131 %                  Creation.
0132 %
0133 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0134
0135 function plo = getDefaultPlist()
0136     plo = plist();
0137 end
0138
0139
```

dropduplicates

PURPOSE ^

DROPDUPPLICATES drops all duplicate samples in time-series AOs.

SYNOPSIS ^

function varargout = dropduplicates(varargin)

DESCRIPTION ^

```
DROPDUPPLICATES drops all duplicate samples in time-series AOs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DROPDUPPLICATES drops all duplicate samples in time-series AOs. Duplicates
are identified by having a two consecutive time stamps
closer than a set tolerance.

CALL:          bs = dropduplicates(as)

INPUTS:        as  - array of analysis objects
                pl  - parameter list (see below)

OUTPUTS:        bs  - array of analysis objects, one for each input

PARAMETER LIST:
                'tol' - specify tolerance (in seconds) for selecting duplicates.
                    [default: 5ms]

M-FILE INFO:   Get information about this methods by calling
                >> ao.getInfo('dropduplicates')

                Get information about a specified set-plist by calling:
                >> ao.getInfo('dropduplicates', 'None')

VERSION:       $Id: dropduplicates.m,v 1.10 2008/09/05 11:05:29 ingo Exp $

HISTORY:       22-05-08 M Hewitson
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [diff](#) DIFF differentiates the data in AO.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```

0001 % DROPDUPLICATES drops all duplicate samples in time-series AOs.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DROPDUPLICATES drops all duplicate samples in time-series AOs. Duplicates
0005 % are identified by having a two consecutive time stamps
0006 % closer than a set tolerance.
0007 %
0008 % CALL:          bs = dropduplicates(as)
0009 %
0010 % INPUTS:        as - array of analysis objects
0011 %                pl - parameter list (see below)
0012 %
0013 % OUTPUTS:       bs - array of analysis objects, one for each input
0014 %
0015 % PARAMETER LIST:
0016 % 'tol' - specify tolerance (in seconds) for selecting duplicates.
0017 % [default: 5ms]
0018 %
0019 % M-FILE INFO: Get information about this methods by calling
0020 % >> ao.getInfo('dropduplicates')
0021 %
0022 % Get information about a specified set-plist by calling:
0023 % >> ao.getInfo('dropduplicates', 'None')
0024 %
0025 % VERSION:       $Id: dropduplicates.m,v 1.10 2008/09/05 11:05:29 ingo Exp $
0026 %
0027 % HISTORY:       22-05-08 M Hewitson
0028 % Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031 %
0032 function varargout = dropduplicates(varargin)
0033 %
0034 % Check if this is a call for parameters
0035 if utils.helper.isinfocall(varargin{:})
0036     varargout{1} = getInfo(varargin{3});
0037     return
0038 end
0039 %
0040 import utils.const.*
0041 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0042 %
0043 % Collect input variable names
0044 in_names = cell(size(varargin));
0045 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0046 %
0047 % Collect all AOs
0048 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0049 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0050 %
0051 % Decide on a deep copy or a modify
0052 bs = copy(as, nargout);
0053 %
0054 % Combine plists
0055 pl = combine(pl, getDefaultPlist);
0056 %
0057 % Get tolerance
0058 tol = find(pl, 'tol');
0059 %
0060 % Get only tsdata AOs
0061 for j=1: numel(bs)
0062     if isa(bs(j).data, 'tsdata')
0063         d = abs(diff(bs(j).data.getX));
0064         idx = find(d<tol);
0065         utils.helper.msg(msg.PROC1, 'found %d duplicate samples', numel(idx));
0066         % Wipe out x samples
0067         if ~isempty(bs(j).data.x)
0068             bs(j).data.x(idx) = [];
0069         end
0070         % Wipe out y samples
0071         bs(j).data.y(idx) = [];
0072         % Add history
0073         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0074         % set name
0075         bs(j).setName(sprintf('dropduplicates(%s)', ao_invars{j}), 'internal');
0076     else
0077         warning('!!! Skipping AO %s - it's not a time-series AO.', ao_invars{j});
0078         bs(j) = [];
0079     end
0080 end
0081 %
0082 % Set output
0083 if nargout > 0
0084     varargout{1} = bs;

```



```
0085     end
0086 end
0087
0088 %-----
0089 % Get Info Object
0090 %-----
0091 function ii = getInfo(varargin)
0092     if nargin == 1 && strcmpi(varargin{1}, 'None')
0093         sets = {};
0094         pl = [];
0095     else
0096         sets = {'Default'};
0097         pl = getDefaultPlist;
0098     end
0099     % Build info object
0100     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id:
dropduplicates.m,v 1.10 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0101 end
0102
0103 %-----
0104 % Get Default Plist
0105 %-----
0106 function pl = getDefaultPlist()
0107     pl = plist('tol', 5e-3);
0108 end
0109
0110
```


dsmean

PURPOSE ^

DSMEAN performs a simple downsampling by taking the mean of every N samples.

SYNOPSIS ^

function varargout = dsmean(varargin)

DESCRIPTION ^

```
DSMEAN performs a simple downsampling by taking the mean of every N samples.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DSMEAN performs a simple downsampling by taking the mean of
              every N samples. The downsample factor (N) is taken as
              round(fs/fsout). The original vector is then truncated to a
              integer number of segments of length N. It is the reshaped
              to N x length(y)/N. Then the mean is taken.

CALL:        b = dsmean(a, pl)

PARAMETERS:   'fsout' - The output sample rate [default: 10]

M-FILE INFO:  Get information about this methods by calling
              >> ao.getInfo('dsmean')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('dsmean', 'None')

VERSION:      $Id: dsmean.m,v 1.9 2008/09/05 11:05:29 ingo Exp $

HISTORY:      20-04-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [mean](#) MEAN computes the mean value of the data in the AO.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DSMEAN performs a simple downsampling by taking the mean of every N samples.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DSMEAN performs a simple downsampling by taking the mean of
```

```

0005 %           every N samples. The downsample factor (N) is taken as
0006 %           round(fs/fsout). The original vector is then truncated to a
0007 %           integer number of segments of length N. It is the reshaped
0008 %           to N x length(y)/N. Then the mean is taken.
0009 %
0010 % CALL:      b = dsmean(a, pl)
0011 %
0012 % PARAMETERS: 'fsout' - The output sample rate [default: 10]
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> ao.getInfo('dsmean')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('dsmean', 'None')
0019 %
0020 % VERSION:    $Id: dsmean.m,v 1.9 2008/09/05 11:05:29 ingo Exp $
0021 %
0022 % HISTORY:     20-04-08 M Hewitson
0023 %               Creation
0024 %
0025 %%%%%%%%%%%
0026
0027 function varargout = dsmean(varargin)
0028
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 import utils.const.*
0036 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0037
0038 % Collect input variable names
0039 in_names = cell(size(varargin));
0040 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0041
0042 % Collect all AOs
0043 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0044 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0045
0046 % Decide on a deep copy or a modify
0047 bs = copy(as, nargout);
0048
0049 % Combine plists
0050 pl = combine(pl, getDefaultPlist);
0051
0052 % Extract necessary parameters
0053 fsout = find(pl, 'fsout');
0054
0055 % Loop over input AOs
0056 for j=1:length(bs)
0057     if ~isa(bs(j).data, 'tsdata')
0058         warning('!!! Can only downsample time-series (tsdata) objects. Skipping AO %s',
0059             ao_invars{j});
0060     else
0061         % downsample factor
0062         dsf = round(bs(j).data.fs/fsout);
0063         if dsf < 1
0064             error('### I can''t downsample - the sample rate is already lower than the
0065 requested.');
```

```
0089     if nargout > 0
0090         varargout{1} = bs;
0091     end
0092 end
0093
0094 %-----
0095 % Get Info Object
0096 %-----
0097 function ii = getInfo(varargin)
0098     if nargin == 1 && strcmpi(varargin{1}, 'None')
0099         sets = {};
0100         pl = [];
0101     else
0102         sets = {'Default'};
0103         pl = getDefaultPlist;
0104     end
0105     % Build info object
0106     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: dsmean.m,v 1.9
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0107 end
0108
0109 %-----
0110 % Get Default Plist
0111 %-----
0112 function pl = getDefaultPlist()
0113     pl = plist('fsout', 10);
0114 end
0115
0116
```

fft

PURPOSE ^

FFT overloads the fft method for Analysis objects.

SYNOPSIS ^

function varargout = fft(varargin)

DESCRIPTION ^

FFT overloads the fft method for Analysis objects.
DESCRIPTION: FFT overloads the fft operator for Analysis objects.
CALL: b = fft(a, pl)
PARAMETERS: 'type' - one or two sided fft [default: 'one']
M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('fft')
Get information about a specified set-plist by calling:
>> ao.getInfo('fft', 'None')
VERSION: \$Id: fft.m,v 1.26 2008/09/05 11:05:29 ingo Exp \$
HISTORY: 12-03-07 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [fft](#) FFT overloads the fft method for Analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [xunits](#) XUNITS Get the data property 'xunits'.

This function is called by:

-  [computeDFT](#) COMPUTEDFT Computes DFT using FFT or Goertzel
-  [fft](#) FFT overloads the fft method for Analysis objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

0001 % FFT overloads the fft method for Analysis objects.
0002 %
0003 %

```

0004 % DESCRIPTION: FFT overloads the fft operator for Analysis objects.
0005 %
0006 % CALL:          b = fft(a, pl)
0007 %
0008 % PARAMETERS:   'type' - one or two sided fft [default: 'one']
0009 %
0010 % M-FILE INFO:  Get information about this methods by calling
0011 %               >> ao.getInfo('fft')
0012 %
0013 %               Get information about a specified set-plist by calling:
0014 %               >> ao.getInfo('fft', 'None')
0015 %
0016 % VERSION:      $Id: fft.m,v 1.26 2008/09/05 11:05:29 ingo Exp $
0017 %
0018 % HISTORY:      12-03-07 M Hewitson
0019 %               Creation
0020 %
0021 %%%%%%%%%%%
0022
0023 function varargout = fft(varargin)
0024
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 import utils.const.*
0032 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0033
0034 % Collect input variable names
0035 in_names = cell(size(varargin));
0036 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0037
0038 % Collect all AOs
0039 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0040 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0041
0042 % Decide on a deep copy or a modify
0043 bs = copy(as, nargin);
0044
0045 % Combine plists
0046 pl = combine(pl, getDefaultPlist);
0047
0048 % two-sided or one?
0049 type = find(pl, 'type');
0050
0051 % Check input analysis object
0052 for j=1:numel(bs)
0053     % Which data type do we have
0054     switch class(bs(j).data)
0055     case {'tsdata', 'cdata', 'xydata'}
0056         % Check we have a sample rate
0057         if strcmp(class(bs(j).data), 'tsdata')
0058             fs = bs(j).data.fs;
0059             fmin = 0;
0060             xunits = 'Hz';
0061         else
0062             warning('!!! Data has no sample rate: setting to %g',
0063 2+length(bs(j).data.y));
0064             fs = 2+length(bs(j).data.y);
0065             fmin = 1;
0066         end
0067         % make FFT of data
0068         nfft = length(bs(j).data.y);
0069         ft = fft(bs(j).data.y);
0070         switch type
0071         case 'one'
0072             ft = ft(1:nfft/2+1);
0073             f = linspace(fmin, fs/2, length(ft));
0074         case 'two'
0075             ft = [ft(nfft/2+2:end) ft(1:nfft/2+1)];
0076             f = linspace(-fs/2, fs/2, length(ft));
0077         otherwise
0078             error('### unknown fft type. ');
0079         end
0080         % Make new fsdata object
0081         fsd = fsdata(f, ft, fs);
0082         fsd.setXunits(xunits);
0083         fsd.setYunits(bs(j).data.yunits);
0084         % make output analysis object
0085         bs(j).data = fsd;
0086         bs(j).setName(sprintf('fft(%s)', ao_invars{j}), 'internal');
0087         % Add history
0088         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0089     otherwise

```

```

0089     error('### You can only fft tsdata, cdata, or xydata AOs.');
```

```

0090     end
0091 end
0092
0093 % Set output
0094 if nargout > 0
0095     varargout{1} = bs;
0096 end
0097 end
0098
0099 %-----
0100 % Get Info Object
0101 %-----
0102 function ii = getInfo(varargin)
0103     if nargin == 1 && strcmpi(varargin{1}, 'None')
0104         sets = {};
0105         pl    = [];
0106     else
0107         sets = {'Default'};
0108         pl    = getDefaultPlist;
0109     end
0110     % Build info object
0111     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: fft.m,v 1.26
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0112 end
0113
0114 %-----
0115 % Get Default Plist
0116 %-----
0117 function pl_default = getDefaultPlist()
0118     pl_default = plist('type', 'one');
0119 end
0120
0121
```

filter

PURPOSE ^

FILTER overrides the filter function for analysis objects.

SYNOPSIS ^

function varargout = filter(varargin)

DESCRIPTION ^

```
FILTER overrides the filter function for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FILTER overrides the filter function for analysis objects.
              Applies the input digital IIR/FIR filter to the input analysis
              object. If the input analysis object contains a
              time-series (tsdata) then the filter is applied using the normal
              recursion algorithm. The output analysis object contains a tsdata
              object.

              If the input analysis object contains a frequency-series (fsdata)
              then the response of the filter is computed and then multiplied
              with the input frequency series. The output analysis object
              contains a frequency series.

CALL:
    >> [b, filt] = filter(a,pl)
    >> [b, filt] = filter(a,filt,pl)
    >> b = filter(a,pl)

INPUTS:
    pl    - a parameter list
    a      - input analysis object

OUTPUTS:
    filt  - a copy of the input filter object with the
            history values filled in.
    b     - output analysis object containing the filtered data.

PARAMETERS: filter - the filter object to use to filter the data

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('filter')

              Get information about a specified set-plist by calling:
    >> ao.getInfo('filter', 'None')

VERSION:    $Id: filter.m,v 1.40 2008/09/05 11:05:29 ingo Exp $

HISTORY:    11-02-07 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [filter](#) FILTER overrides the filter function for analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

-  [filter](#) FILTER overrides the filter function for analysis objects.
-  [firwhiten](#) FIRWHITEN whitens the input time-series by building an FIR whitening filter.
-  [gapfilling](#) GAPFILLING fills possible gaps in data.

SUBFUNCTIONS [^]

 function ii = getInfo(varargin)
 function pl_default = getDefaultPlist()

SOURCE CODE [^]

```

0001 % FILTER overrides the filter function for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FILTER overrides the filter function for analysis objects.
0005 %               Applies the input digital IIR/FIR filter to the input analysis
0006 %               object. If the input analysis object contains a
0007 %               time-series (tsdata) then the filter is applied using the normal
0008 %               recursion algorithm. The output analysis object contains a tsdata
0009 %               object.
0010 %
0011 %               If the input analysis object contains a frequency-series (fsdata)
0012 %               then the response of the filter is computed and then multiplied
0013 %               with the input frequency series. The output analysis object
0014 %               contains a frequency series.
0015 %
0016 % CALL:         >> [b, filt] = filter(a,pl)
0017 %               >> [b, filt] = filter(a,filt,pl)
0018 %               >> b = filter(a,pl)
0019 %
0020 % INPUTS:       pl - a parameter list
0021 %               a  - input analysis object
0022 %
0023 % OUTPUTS:      filt - a copy of the input filter object with the
0024 %                   history values filled in.
0025 %               b   - output analysis object containing the filtered data.
0026 %
0027 % PARAMETERS:   filter - the filter object to use to filter the data
0028 %
0029 % M-FILE INFO:  Get information about this methods by calling
0030 %               >> ao.getInfo('filter')
0031 %
0032 %               Get information about a specified set-plist by calling:
0033 %               >> ao.getInfo('filter', 'None')
0034 %
0035 % VERSION:      $Id: filter.m,v 1.40 2008/09/05 11:05:29 ingo Exp $
0036 %
0037 % HISTORY:      11-02-07 M Hewitson
0038 %               Creation
0039 %
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041 %
0042 function varargout = filter(varargin)
0043 %
0044 % Check if this is a call for parameters
0045 if utils.helper.isinfocall(varargin{:})
0046     varargout{1} = getInfo(varargin{3});
0047     return
0048 end
0049 %
0050 import utils.const.*
0051 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0052 %
0053 % Collect input variable names
0054 in_names = cell(size(varargin));
0055 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0056 %
0057 % Collect all AOs and plists
0058 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0059 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0060 [fobj, f_invars] = utils.helper.collect_objects(varargin(:), 'ltpda_filter',
in_names);
0061 %
0062 % Make copies or handles to inputs
0063 bs = copy(as, nargout);
0064 %
0065 % combine plists
0066 pl = combine(pl, getDefaultPlist());
0067 %
0068 if isempty(fobj)
0069     fobj = find(pl, 'filter');
0070 end
0071 %
0072 % check inputs

```



```

0073 if ~isa(fobj, 'miir') && ~isa(fobj, 'mfir')
0074     error('### the filter input should be an miir/mfir object.');
```

0075 end

0076

0077 % Now copy the filter so we can change it

0078 fobj = copy(fobj, 1);

0079

0080 for j=1:numel(bs)

0081 %----- Time-series filter

0082 if isa(bs(j).data, 'tsdata')

0083 % get input data

0084 if isa(fobj, 'mfir')

0085 % apply filter

0086 utils.helper.msg(msg.PROC1, 'filtering with FIR filter');

0087 [bs(j).data.y, Zf] = filter(fobj.a, 1, bs(j).data.y, fobj.histout);

0088 fobj.setHistout(Zf);

0089 % remove group delay

0090 if isempty(find(pl, 'gdoff'))

0091 gd = floor(fobj.gd);

0092 bs(j).data.setX(bs(j).data.getX(1:end-gd));

0093 bs(j).data.setY(bs(j).data.y(1+gd:end));

0094 bs(j).data.collapseX;

0095 end

0096 else %if isa(fobj, 'miir')

0097 if bs(j).data.fs ~= fobj.fs

0098 warning('!!! Filter is designed for a different sample rate of data.');

0099 % Adjust/redesign if this is a standard filter

0100 fobj = fobj.redesign(bs(j).data.fs);

0101 end

0102 utils.helper.msg(msg.PROC1, 'filtering with IIR filter');

0103 [bs(j).data.y, Zf] = filter(fobj.a, fobj.b, bs(j).data.y, fobj.histout);

0104 fobj.setHistout(Zf);

0105 end

0106 % add history

0107 bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);

0108 % name for this object

0109 bs(j).setName(sprintf('%s(%s)', fobj.name, ao_invars{j}), 'internal');

0110 elseif isa(d, 'fsdata')

0111 %----- Frequency-series filter

0112 error('### I don''t work yet. Please code me up.');

0113 else

0114 error('### unknown data type.');

0115 end

0116 end

0117

0118

0119 % Set outputs

0120 if nargout == 1

0121 varargout{1} = bs;

0122 elseif nargout == 2

0123 varargout{1} = bs;

0124 varargout{2} = fobj;

0125 elseif nargout > 2

0126 error('### wrong number of output arguments.');

0127 end

0128 end

0129

0130 %-----

0131 % Get Info Object

0132 %-----

0133 function ii = getInfo(varargin)

0134 if nargin == 1 && strcmpi(varargin{1}, 'None')

0135 sets = {};

0136 pls = [];

0137 else

0138 sets = {'Default'};

0139 pls = getDefaultPlist;

0140 end

0141 % Build info object

0142 ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '\$Id: filter.m,v 1.40

2008/09/05 11:05:29 ingo Exp \$', sets, pls);

0143 end

0144

0145 %-----

0146 % Get Default Plist

0147 %-----

0148 function pl_default = getDefaultPlist()

0149 pl_default = plist('filter', '', 'gdoff', []);

0150 end

0151

filtfilt

PURPOSE ^

FILTFILT overrides the filtfilt function for analysis objects.

SYNOPSIS ^

function varargout = filtfilt(varargin)

DESCRIPTION ^

```
FILTFILT overrides the filtfilt function for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FILTFILT overrides the filtfilt function for analysis objects.
              Applies the input digital IIR filter to the input analysis object
              forwards and backwards. If the input analysis object contains a
              time-series (tsdata) then the filter is applied using the normal
              recursion algorithm. The output analysis object contains a tsdata
              object.

              If the input analysis object contains a frequency-series (fsdata)
              then the response of the filter is computed and then multiplied
              with the input frequency series. The output analysis object
              contains a frequency series.

CALL:         >> [b, filt] = filtfilt(a,pl)
              >> b = filtfilt(a,pl)

INPUTS:       pl    - a parameter list
              a      - input analysis object

OUTPUTS:       filt  - a copy of the input filter object with the
                    history values filled in.
              b      - output analysis object containing the filtered data.

PARAMETERS:    filter - the filter object to use to filter the data

M-FILE INFO:   Get information about this methods by calling
              >> ao.getInfo('filtfilt')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('filtfilt', 'None')

VERSION:       $Id: filtfilt.m,v 1.27 2008/09/05 11:05:29 ingo Exp $

HISTORY:       11-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.

This function is called by:

-  [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
-  [gapfilling](#) GAPFILLING fills possible gaps in data.

 **spikecleaning** spikecleaning detects and corrects possible spikes in analysis objects

SUBFUNCTIONS [^]

 **function ii = getInfo(varargin)**
 **function pl_default = getDefaultPlist()**

SOURCE CODE [^]

```

0001 % FILTFILT overrides the filtfilt function for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FILTFILT overrides the filtfilt function for analysis objects.
0005 %               Applies the input digital IIR filter to the input analysis object
0006 %               forwards and backwards. If the input analysis object contains a
0007 %               time-series (tsdata) then the filter is applied using the normal
0008 %               recursion algorithm. The output analysis object contains a tsdata
0009 %               object.
0010 %
0011 %               If the input analysis object contains a frequency-series (fsdata)
0012 %               then the response of the filter is computed and then multiplied
0013 %               with the input frequency series. The output analysis object
0014 %               contains a frequency series.
0015 %
0016 % CALL:         >> [b, filt] = filtfilt(a,pl)
0017 %               >> b = filtfilt(a,pl)
0018 %
0019 % INPUTS:       pl    - a parameter list
0020 %               a     - input analysis object
0021 %
0022 % OUTPUTS:      filt  - a copy of the input filter object with the
0023 %                       history values filled in.
0024 %               b     - output analysis object containing the filtered data.
0025 %
0026 % PARAMETERS:   filter - the filter object to use to filter the data
0027 %
0028 % M-FILE INFO:  Get information about this methods by calling
0029 %               >> ao.getInfo('filtfilt')
0030 %
0031 %               Get information about a specified set-plist by calling:
0032 %               >> ao.getInfo('filtfilt', 'None')
0033 %
0034 % VERSION:      $Id: filtfilt.m,v 1.27 2008/09/05 11:05:29 ingo Exp $
0035 %
0036 % HISTORY:      11-02-07 M Hewitson
0037 %               Creation
0038 %
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040
0041 function varargout = filtfilt(varargin)
0042
0043 % Check if this is a call for parameters
0044 if utils.helper.isinfocall(varargin{:})
0045     varargout{1} = getInfo(varargin{3});
0046     return
0047 end
0048
0049 import utils.const.*
0050 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0051
0052 % Collect input variable names
0053 in_names = cell(size(varargin));
0054 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0055
0056 % Collect all AOs and plists
0057 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0058 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0059 [fobj, f_invars] = utils.helper.collect_objects(varargin(:), 'ltpda_filter',
in_names);
0060
0061 % Make copies or handles to inputs
0062 bs    = copy(as, nargout);
0063
0064 % combine plists
0065 pl = combine(pl, getDefaultPlist());
0066
0067 if isempty(fobj)
0068     fobj = find(pl, 'filter');
0069     f_invars{1} = class(fobj);
0070 end

```

```

0071
0072 % check inputs
0073 if ~isa(fobj, 'miir') && ~isa(fobj, 'mfir')
0074     error('### the filter input should be an miir/mfir object.');
```

```

0075 end
0076
0077 % Now copy the filter so we can change it
0078 fobj = copy(fobj, 1);
0079
0080 % Loop over AOs
0081 for j=1:numel(bs)
0082     %----- Time-series filter
0083     if isa(bs(j).data, 'tsdata')
0084         fs = bs(j).data.fs;
0085         if fs ~= fobj.fs
0086             warning('!!! Filter is designed for a different sample rate of data.');
```

```

0087             % Adjust/redesign if this is a standard filter
0088             fobj = redesign(fobj, fs);
0089         end
0090         % apply filter
0091         bs(j).data.y = filtfilt(fobj.a, fobj.b, bs(j).data.y);
0092         % add history
0093         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0094         % name for this object
0095         bs(j).setName(sprintf('%s(%s)', f_invars{1}, ao_invars{j}), 'internal');
```

```

0096     elseif isa(d, 'fsdata')
0097         %----- Frequency-series filter
0098         error('### I don''t work yet. Please code me up.');
```

```

0099     else
0100         error('### unknown data type.');
```

```

0101     end
0102 end
0103
0104 % Set outputs
0105 if nargout == 1
0106     varargout{1} = bs;
0107 elseif nargout == 2
0108     varargout{1} = bs;
0109     varargout{2} = fobj;
0110 elseif nargout > 2
0111     error('### wrong number of output arguments.');
```

```

0112 end
0113 end
0114
0115 %-----
0116 % Get Info Object
0117 %-----
0118 function ii = getInfo(varargin)
0119     if nargin == 1 && strcmpi(varargin{1}, 'None')
0120         sets = {};
0121         pls = [];
0122     else
0123         sets = {'Default'};
0124         pls = getDefaultPlist;
0125     end
0126     % Build info object
0127     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: filtfilt.m,v
1.27 2008/09/05 11:05:29 ingo Exp $', sets, pls);
0128 end
0129
0130 %-----
0131 % Get Default Plist
0132 %-----
0133 function pl_default = getDefaultPlist()
0134     pl_default = plist('filter', '');
0135 end
0136
0137

```

firwhiten

PURPOSE ^

FIRWHITEN whitens the input time-series by building an FIR whitening filter.

SYNOPSIS ^

function varargout = firwhiten(varargin)

DESCRIPTION ^

FIRWHITEN whitens the input time-series by building an FIR whitening filter.
%%%

DESCRIPTION: FIRWHITEN whitens the input time-series by building an FIR whitening filter. The algorithm ultimately uses fir2() to build the whitening filter.

ALGORITHM:

- 1) Make ASD of time-series
- 2) Perform running median to get noise-floor estimate (ao/smoothen)
- 3) Invert noise-floor estimate
- 4) Call mfir() on noise-floor estimate to produce whitening filter
- 5) Filter data

CALL:

```
b = firwhiten(a, pl) % returns whitened time-series AOs
[b, filts] = firwhiten(a, pl) % returns the mfir filters used
[b, filts, nfs] = firwhiten(a, pl) % returns the noise-floor
                                     % estimates as fsdata AOs
```

PARAMETERS:

- 'Ntaps' - the length of the FIR filter to build [default: 256].
- 'FIRwin' - the window to use in the filter design. Pass a specwin object of the desired type and of any length. [default: Hanning]

parameters passed to ltpda_pwelch()

- 'Nfft' - The number of points in the FFT used to estimate the power spectrum. [default: Ndata/4]
- 'Win' - Spectral window used in spectral estimation. [default: Kaiser -150dB]
- 'Order' - order of segment detrending:
 - 1 - no detrending
 - 0 - subtract mean [default]
 - 1 - subtract linear fit
 - N - subtract fit of polynomial, order N

(Segment overlap is taken from the window function.)

parameters passed to ltpda_nfest()

- 'bw' - The bandwidth of the running median filter used to estimate the noise-floor. [default: 20 samples]
- 'hc' - The cutoff used to reject outliers (0-1) [default: 0.8]

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('firwhiten')

Get information about a specified set-plist by calling:
>> ao.getInfo('firwhiten', 'None')

VERSION: \$Id: firwhiten.m,v 1.10 2008/09/08 08:30:12 hewitson Exp \$

HISTORY: 21-04-08 M Hewitson
Creation

%%%

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  **copy** COPY makes a (deep) copy of the input AOs.
-  **filter** FILTER overrides the filter function for analysis objects.
-  **find** FIND particular samples that satisfy the input query and return a new AO.
-  **psd** PSD makes power spectral density estimates of the time-series objects
-  **smoother** SMOOTHER smooths a given series of data points using the specified method.

This function is called by:

SUBFUNCTIONS [^]

-  **function ii = getInfo(varargin)**
-  **function pl_default = getDefaultPlist()**

SOURCE CODE [^]

```

0001 % FIRWHITEN whitens the input time-series by building an FIR whitening filter.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FIRWHITEN whitens the input time-series by building an FIR
0005 %                whitening filter. The algorithm ultimately uses fir2() to
0006 %                build the whitening filter.
0007 %
0008 % ALGORITHM:
0009 %           1) Make ASD of time-series
0010 %           2) Perform running median to get noise-floor estimate (ao/smoother)
0011 %           3) Invert noise-floor estimate
0012 %           4) Call mfir() on noise-floor estimate to produce whitening filter
0013 %           5) Filter data
0014 %
0015 % CALL:
0016 %           b = firwhiten(a, pl) % returns whitened time-series AOs
0017 %           [b, filts] = firwhiten(a, pl) % returns the mfir filters used
0018 %           [b, filts, nfs] = firwhiten(a, pl) % returns the noise-floor
0019 %                                           % estimates as fsdata AOs
0020 % PARAMETERS:
0021 %
0022 %           'Ntaps' - the length of the FIR filter to build [default: 256].
0023 %           'FIRwin' - the window to use in the filter design. Pass a
0024 %                      specwin object of the desired type and of any length.
0025 %                      [default: Hanning]
0026 %
0027 % parameters passed to ltpda_pwelch()
0028 %
0029 %           'Nfft' - The number of points in the FFT used to estimate
0030 %                   the power spectrum.
0031 %                   [default: Ndata/4]
0032 %           'Win' - Spectral window used in spectral estimation.
0033 %                   [default: Kaiser -150dB]
0034 %           'Order' - order of segment detrending:
0035 %                   -1 - no detrending
0036 %                   0 - subtract mean [default]
0037 %                   1 - subtract linear fit
0038 %                   N - subtract fit of polynomial, order N
0039 %
0040 % (Segment overlap is taken from the window function.)
0041 %
0042 % parameters passed to ltpda_nfest()
0043 %
0044 %           'bw' - The bandwidth of the running median filter used to
0045 %                 estimate the noise-floor.
0046 %                 [default: 20 samples]
0047 %           'hc' - The cutoff used to reject outliers (0-1)
0048 %                 [default: 0.8]
0049 %
0050 % M-FILE INFO: Get information about this methods by calling
0051 %               >> ao.getInfo('firwhiten')
0052 %
0053 %               Get information about a specified set-plist by calling:
0054 %               >> ao.getInfo('firwhiten', 'None')
0055 %

```



```

0056 % VERSION:      $Id: firwhiten.m,v 1.10 2008/09/08 08:30:12 hewitson Exp $
0057 %
0058 % HISTORY: 21-04-08 M Hewitson
0059 %      Creation
0060 %
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062 %
0063 function varargout = firwhiten(varargin)
0064
0065 % Check if this is a call for parameters
0066 if utils.helper.isinfocall(varargin{:})
0067     varargout{1} = getInfo(varargin{3});
0068     return
0069 end
0070
0071 import utils.const.*
0072 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0073
0074 % Collect input variable names
0075 in_names = cell(size(varargin));
0076 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0077
0078 % Collect all AOs and plists
0079 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0080 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0081
0082 % Decide on a deep copy or a modify
0083 bs = copy(as, nargout);
0084 inhists = copy([as.hist],1);
0085
0086 % combine plists
0087 pl = combine(pl, getDefaultPlist());
0088
0089 % Extract necessary parameters
0090 infft = find(pl, 'Nfft');
0091 bw = find(pl, 'bw');
0092 hc = find(pl, 'hc');
0093 swin = find(pl, 'win');
0094 order = find(pl, 'order');
0095 fwin = find(pl, 'FIRwin');
0096 Ntaps = find(pl, 'Ntaps');
0097
0098 % Loop over input AOs
0099 filts = [];
0100 nfs = [];
0101
0102 for j=1:length(bs)
0103     if ~isa(bs(j).data, 'tsdata')
0104         warning('!!! %s expects ao/tsdata objects. Skipping AO %s', mfilename,
0105             ao_invars{j});
0106         bs(j) = [];
0107     else
0108         % get Nfft
0109         if infft < 0
0110             Nfft = length(bs(j).data.y);
0111         else
0112             Nfft = infft;
0113         end
0114         utils.helper.msg(msg.PROC1, 'building spectrum');
0115         % Make spectrum
0116         axx = psd(bs(j), plist('Nfft', Nfft, 'Win', swin, 'Order', order, 'Scale',
0117             'ASD'));
0118         % make noise floor estimate
0119         utils.helper.msg(msg.PROC1, 'estimating noise-floor');
0120         nxx = smoother(axx, plist('width', bw, 'hc', hc));
0121         % collect noise-floor estimates for output
0122         nfs = [nfs nxx];
0123         % invert and make weights
0124         w = 1./nxx;
0125         % Make mfir object
0126         utils.helper.msg(msg.PROC1, 'building filter');
0127         ff = mfir(w, plist('Win', fwin, 'N', Ntaps));
0128         % collect filters for output
0129         filts = [filts ff];
0130         % Filter data
0131         utils.helper.msg(msg.PROC1, 'filter data');
0132         filter(bs(j), ff);
0133         % Set name
0134         bs(j).setName(sprintf('firwhiten(%s)', ao_invars{j}), 'internal');
0135         % add history
0136         bs(j).addHistory(getInfo, pl, ao_invars(j), inhists(j));
0137     end
0138 end
0139
0140 % Set outputs
0141 if nargout > 0

```

```

0140     varargout{1} = bs;
0141     end
0142     if nargout > 1
0143         varargout{2} = filts;
0144     end
0145     if nargout > 2
0146         varargout{3} = nfs;
0147     end
0148 end
0149
0150 %-----
0151 % Get Info Object
0152 %-----
0153 function ii = getInfo(varargin)
0154     if nargin == 1 && strcmpi(varargin{1}, 'None')
0155         sets = {};
0156         pl = [];
0157     else
0158         sets = {'Default'};
0159         pl = getDefaultPlist;
0160     end
0161     % Build info object
0162     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: firwhiten.m,v
1.10 2008/09/08 08:30:12 hewitson Exp $', sets, pl);
0163 end
0164
0165 %-----
0166 % Get Default Plist
0167 %-----
0168 function pl_default = getDefaultPlist()
0169     pl_default = plist(...
0170         'Nfft', -1, ...
0171         'bw', 20, ...
0172         'hc', 0.8, ...
0173         'win', specwin('Kaiser', 10, 150), ...
0174         'order', 0, ...
0175         'FIRwin', specwin('Hanning', 10), ...
0176         'Ntaps', 256);
0177 end
0178
0179

```


fixfs

PURPOSE ^

FIXFS resamples the input time-series to have a fixed sample rate.

SYNOPSIS ^

function varargout = fixfs(varargin)

DESCRIPTION ^

```
FIXFS resamples the input time-series to have a fixed sample rate.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FIXFS resamples the input time-series to have a fixed sample rate.

The new sampling grid is computed from the specified sample rate. If no
sample rate is specified, the target is taken from a fit to the input tsdata
object. The new sampling grid starts at the time returned from the fit
(unless specified) and contains the same number of points or spans the
same time as specified.

>> bs = fixfs(as)

Inputs:
  as - array of analysis objects
  pl - parameter list (see below)

Outputs:
  bs - array of analysis objects, one for each input

Parameter list:
  'fs' - specify the target sample rate. Either a single value
        for all input time-series, or a vector of values, one
        for each input. To take a fitted value from the data,
        specify a sample rate of -1.
        e.g.: fs = [1 2 -1 4] to work on 4 input time-series
        [default: take from data]

  'Method' - Choose the behaviour
              'Time' - new data span the sam time [default]
              'Samples' - new data preserves number of samples

  't0' - specify start time of the sample grid. Set a single
        value or one per input time-series. Use a value of -1
        to take a fitted value from the data.

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('fixfs')

Get information about a specified set-plist by calling:
>> ao.getInfo('fixfs', 'None')

M Hewitson 26-05-08

$Id: fixfs.m,v 1.11 2008/09/05 11:05:29 ingo Exp $
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [interp](#) INTERP interpolate the values in the input AO(s) at new values.

 **t0** T0 Get the data property 't0'.

This function is called by:

 **consolidate** CONSOLIDATE resamples all input AOs onto the same time grid.

SUBFUNCTIONS [^]

 **function ii = getInfo(varargin)**

 **function pl_default = getDefaultPlist()**

SOURCE CODE [^]

```

0001 % FIXFS resamples the input time-series to have a fixed sample rate.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FIXFS resamples the input time-series to have a fixed sample rate.
0005 %
0006 % The new sampling grid is computed from the specified sample rate. If no
0007 % sample rate is specified, the target is taken from a fit to the input tsdata
0008 % object. The new sampling grid starts at the time returned from the fit
0009 % (unless specified) and contains the same number of points or spans the
0010 % same time as specified.
0011 %
0012 %     >> bs = fixfs(as)
0013 %
0014 % Inputs:
0015 %     as - array of analysis objects
0016 %     pl - parameter list (see below)
0017 %
0018 % Outputs:
0019 %     bs - array of analysis objects, one for each input
0020 %
0021 % Parameter list:
0022 %     'fs' - specify the target sample rate. Either a single value
0023 %           for all input time-series, or a vector of values, one
0024 %           for each input. To take a fitted value from the data,
0025 %           specify a sample rate of -1.
0026 %           e.g.: fs = [1 2 -1 4] to work on 4 input time-series
0027 %           [default: take from data]
0028 %
0029 %     'Method' - Choose the behaviour
0030 %                 'Time' - new data span the sam time [default]
0031 %                 'Samples' - new data preserves number of samples
0032 %
0033 %     't0' - specify start time of the sample grid. Set a single
0034 %           value or one per input time-series. Use a value of -1
0035 %           to take a fitted value from the data.
0036 %
0037 % M-FILE INFO: Get information about this methods by calling
0038 %     >> ao.getInfo('fixfs')
0039 %
0040 %           Get information about a specified set-plist by calling:
0041 %     >> ao.getInfo('fixfs', 'None')
0042 %
0043 % M Hewitson 26-05-08
0044 %
0045 % $Id: fixfs.m,v 1.11 2008/09/05 11:05:29 ingo Exp $
0046 %
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048 %
0049 function varargout = fixfs(varargin)
0050 %
0051 % Check if this is a call for parameters
0052 if utils.helper.isinfocall(varargin{:})
0053     varargout{1} = getInfo(varargin{3});
0054     return
0055 end
0056 %
0057 import utils.const.*
0058 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0059 %
0060 % Collect input variable names
0061 in_names = cell(size(varargin));
0062 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0063 %
0064 % Collect all AOs and plists
0065 [as, ao invars] = utils.helper.collect objects(varargin(:), 'ao', in names);

```

```

0066 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0067
0068 % Decide on a deep copy or a modify
0069 bs = copy(as, nargout);
0070
0071 % combine plists
0072 pl = combine(pl, getDefaultPlist());
0073
0074 % Get fs and t0
0075 t0s = find(pl, 't0');
0076 fss = find(pl, 'fs');
0077 method = find(pl, 'method');
0078
0079 if numel(fss) ~= 1 && numel(fss) < na
0080     error('### Please specify either a no sample rate, a single sample rate, or one for
each input time-series.');
```

```

0081 end
0082
0083 % Get only tsdata A0s
0084 for j=1:numel(bs)
0085     if isa(bs(j).data, 'tsdata')
0086         % record input hist
0087         hin = bs(j).hist;
0088         utils.helper.msg(msg.PROC1, 'fixing A0: %s', bs(j).name);
0089         %----- Fit sample rate and t0
0090         [ffs, ft0] = tsdata.fitfs(bs(j).data.getX);
0091         %----- Get target sample rate
0092         if numel(fss) > 1
0093             fs = fss(j);
0094         else
0095             fs = fss;
0096         end
0097         if fs < 0
0098             utils.helper.msg(msg.PROC1, 'using sample rate from fit: %f', ffs);
0099             fs = ffs;
0100         end
0101         %----- Get target start time
0102         if numel(t0s) > 1
0103             t0 = t0s(j);
0104         else
0105             t0 = t0s;
0106         end
0107         if t0 < 0
0108             utils.helper.msg(msg.PROC1, 'using start time from fit: %f', ft0);
0109             t0 = ft0;
0110         end
0111         t0 = t0 + bs(j).data.getX(1);
0112         %----- Compute new grid
0113         switch method
0114             case 'Samples'
0115                 N = length(bs(j).data.y);
0116                 t = linspace(t0, (N-1)/fs, N);
0117             case 'Time'
0118                 Nsecs = bs(j).data.nsecs;
0119                 t = t0 + [0:1/fs:Nsecs-1/fs].';
0120             otherwise
0121                 error('### Unknown interpolation method. Do you want to preserve data
duration or number of samples?');
```

```

0122         end
0123         %----- Interpolate
0124         interp(bs(j), plist('vertices', t));
0125         % Add history
0126         bs(j).addHistory(getInfo, pl, ao_invars(j), hin);
0127         % Set name
0128         bs(j).setName(sprintf('%s(%s)', mfilename, ao_invars{j}), 'internal');
0129     else
0130         warning('!!! Skipping A0 %s - it's not a time-series A0.', ao_invars{j});
0131         bs(j) = [];
0132     end
0133 end
0134
0135 % Set output
0136 if nargout > 0
0137     varargout{1} = bs;
0138 end
0139 end
0140
0141 %-----
0142 % Get Info Object
0143 %-----
0144 function ii = getInfo(varargin)
0145     if nargin == 1 && strcmpi(varargin{1}, 'None')
0146         sets = {};
0147         pl = [];
0148     else
0149         sets = {'Default'};
```

```
0150     pl     = getDefaultPlist;
0151     end
0152     % Build info object
0153     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: fixfs.m,v 1.11
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0154 end
0155
0156 %-----
0157 % Get Default Plist
0158 %-----
0159 function pl_default = getDefaultPlist()
0160     pl_default = plist('fs', -1, 't0', -1, 'Method', 'Time');
0161 end
0162
0163
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fngen

PURPOSE ^

FNGEN creates an arbitrarily long time-series based on the input PSD.

SYNOPSIS ^

function varargout = fngen(varargin)

DESCRIPTION ^

```
FNGEN creates an arbitrarily long time-series based on the input PSD.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FNGEN creates an arbitrarily long time-series based on the input PSD.

CALL:      b = fngen(axx, pl)

PARAMETERS:
    'Nsecs' - The number of seconds to produce
               [default: inverse of PSD length]
    'Win'    - The spectral window to use for blending segments
               [default: Kaiser -150dB]

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('fngen')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('fngen', 'None')

VERSION:     $Id: fngen.m,v 1.17 2008/09/05 11:05:29 ingo Exp $

HISTORY:     12-03-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [ifft](#) IFFT overloads the ifft operator for Analysis objects.
-  [sqrt](#) SQRT computes the square root of the data in the AO.
-  [x](#) X Get the data property 'x'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % FNGEN creates an arbitrarily long time-series based on the input PSD.
```

```

0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FNGEN creates an arbitrarily long time-series based on the input PSD.
0005 %
0006 % CALL:          b = fngen(axx, pl)
0007 %
0008 % PARAMETERS:
0009 %       'Nsecs'   - The number of seconds to produce
0010 %                   [default: inverse of PSD length]
0011 %       'Win'     - The spectral window to use for blending segments
0012 %                   [default: Kaiser -150dB]
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> ao.getInfo('fngen')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('fngen', 'None')
0019 %
0020 % VERSION:       $Id: fngen.m,v 1.17 2008/09/05 11:05:29 ingo Exp $
0021 %
0022 % HISTORY:       12-03-07 M Hewitson
0023 %                 Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 function varargout = fngen(varargin)
0028
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 import utils.const.*
0036 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0037
0038 % Collect input variable names
0039 in_names = cell(size(varargin));
0040 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0041
0042 % Collect all AOs and plists
0043 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0044 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0045
0046 if nargout == 0
0047     error('### fngen cannot be used as a modifier. Please give an output variable.');
```

```

0087 % Prepare for generation
0088 rphi = zeros(N,1); % Empty vector for random phases
0089 xs = zeros(fs*(cNsecs+segLen), 1); % Large empty vector for new time-series
0090 e1 = 1; e2 = segLen*fs; % Indices into large vector
0091 step = round(segLen*fs*0lap); % step size between each new segment
0092 lxs = length(xs);
0093
0094 % Loop over segments
0095 for s=1:Nsegs
0096 % Generate random phase vector
0097 rphi(2:N/2) = pi*rand(1,N/2-1); % First half
0098 rphi(N/2+1) = pi*round(rand); % mid point
0099 rphi(N/2+2:N) = -rphi(N/2:-1:2); % reflected half
0100 %---- Compute Fourier amplitudes
0101 % Use chi^2 distribution to randomize amplitudes.
0102 % - from Percival and Walden: S_est = S.*chi2rnd(2)/2
0103 % so A_est = A.*sqrt(chi2rnd(2)/2)
0104 % Here we take the measured input data to be a good estimate of
0105 % the underlying power spectrum
0106 X = (Ak.*sqrt(chi2rnd(2)/2)) .*exp(1i.*rphi);
0107 % Inverse FFT
0108 x = ifft(X, 'symmetric');
0109 % overlap the segments
0110 xs(e1:e2) = xs(e1:e2) + win.*x;
0111 % increase step
0112 e1 = e1 + step;
0113 e2 = e2 + step;
0114 if e2>lxs
0115 break
0116 end
0117 end
0118 % Make ao from the segment of data we want
0119 e1 = fs*segLen/2;
0120 e2 = fs*(Nsecs+segLen/2)-1;
0121 b = ao(tsddata(xs(e1:e2), fs));
0122 b.setName(sprintf('fngen(%s)', ao_invars{j}), 'internal');
0123 b.setXunits('s', 'internal');
0124 % Add history
0125 b.addHistory(getInfo, pl, ao_invars(j), as(j).hist);
0126 % Add to outputs
0127 bs = [bs b];
0128 end
0129 end
0130
0131 % Set outputs
0132 varargout{1} = bs;
0133 end
0134
0135 %-----
0136 % Get Info Object
0137 %-----
0138 function ii = getInfo(varargin)
0139 if nargin == 1 && strcmpi(varargin{1}, 'None')
0140 sets = {};
0141 pl = [];
0142 else
0143 sets = {'Default'};
0144 pl = getDefaultPlist;
0145 end
0146 % Build info object
0147 ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: fngen.m,v 1.17
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0148 end
0149
0150 %-----
0151 % Get Default Plist
0152 %-----
0153 function pl_default = getDefaultPlist()
0154 pl_default = plist('Nsecs', -1, 'Win', getappdata(0,
'ltppda_default_spectral_window'));
0155 end
0156

```


gapfilling

PURPOSE ^

GAPFILLING fills possible gaps in data.

SYNOPSIS ^

function varargout = gapfilling(varargin)

DESCRIPTION ^

```
GAPFILLING fills possible gaps in data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: GAPFILLING interpolated data between two data
              segments. This function might be useful for possible
              gaps or corrupted data. Two different types of
              interpolating are available: linear and spline, the latter
              results in a smoother curve between the two data segments.

CALL:        b = gapfilling(a1, a2, pl)

INPUTS:      a1 - data segment previous to the gap
              a2 - data segment posterior to the gap
              pl - parameter list

OUTPUTS:     b - data segment containing a1, a2 and the filled data
              segment, i.e., b=[a1 dataare_filled a2].

PARAMETERES: 'method' - method used to interpolate data between a1 and a2.
              Two options can be used: 'linear' and 'spline'.
              Default values is 'linear'.
              'addnoise' - noise can be added to the interpolated data.
              This noise is defined as random variable with
              zero mean and variance equal to the high-frequency
              noise if a1. 'Yes' adds noise. Default value
              is 'no'.

VERSION:     $Id: gapfilling.m,v 1.5 2008/09/05 11:05:29 ingo Exp $

HISTORY:     28-12-2007 J Sanjuan
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [diff](#) DIFF differentiates the data in AO.
-  [filter](#) FILTER overrides the filter function for analysis objects.
-  [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [mean](#) MEAN computes the mean value of the data in the AO.
-  [std](#) STD computes the standard deviation of the data in the AO.
-  [x](#) X Get the data property 'x'.

This function is called by:

SUBFUNCTIONS ^

 `function plf = getlpFilter(x)`
 `function phf = gethpFilter(x)`
 `function ii = getInfo(varargin)`
 `function plo = getDefaultPlist()`

SOURCE CODE ^

```

0001 % GAPFILLING fills possible gaps in data.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: GAPFILLING interpolated data between two data
0005 %               segments. This function might be useful for possible
0006 %               gaps or corrupted data. Two different types of
0007 %               interpolating are available: linear and spline, the latter
0008 %               results in a smoother curve between the two data segments.
0009 %
0010 % CALL:         b = gapfilling(a1, a2, pl)
0011 %
0012 % INPUTS:       a1 - data segment previous to the gap
0013 %               a2 - data segment posterior to the gap
0014 %               pl - parameter list
0015 %
0016 % OUTPUTS:      b - data segment containing a1, a2 and the filled data
0017 %               segment, i.e., b=[a1 dataare_filled a2].
0018 %
0019 % PARAMETERES:  'method' - method used to interpolate data between a1 and a2.
0020 %               Two options can be used: 'linear' and 'spline'.
0021 %               Default values is 'linear'.
0022 %               'addnoise' - noise can be added to the interpolated data.
0023 %               This noise is defined as random variable with
0024 %               zero mean and variance equal to the high-frequency
0025 %               noise if a1. 'Yes' adds noise. Default value
0026 %               is 'no'.
0027 %
0028 % VERSION:      $Id: gapfilling.m,v 1.5 2008/09/05 11:05:29 ingo Exp $
0029 %
0030 % HISTORY:      28-12-2007 J Sanjuan
0031 %               Creation
0032 %
0033 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034
0035 function varargout = gapfilling(varargin)
0036
0037 %% Check if this is a call for parameters
0038 if utils.helper.isinfocall(varargin{:})
0039     varargout{1} = getInfo(varargin{3});
0040     return
0041 end
0042
0043 if nargin == 0
0044     error('### cat cannot be used as a modifier. Please give an output variable.');
```

```

data
0075
0076 %-----
0077 % Get data type from the first A0
0078 a1 = as(1).data;
0079 dinfo1 = whos('a1');
0080 dtype1 = dinfo1.class;
0081
0082 % Get data type from the second A0
0083 a2 = as(2).data;
0084 dinfo2 = whos('a2');
0085 dtype2 = dinfo2.class;
0086
0087 if a1.fs ~= a2.fs
0088     warning('Sampling frequencies of the two A0s are different. The sampling frequency
of the first A0 will be used to reconstruct the gap.')
0089 end
0090
0091 % Different units error
0092 if a1.xunits ~= a2.xunits
0093     error('!!! Data has different X units !!!');
0094 end
0095
0096 if a1.yunits ~= a2.yunits
0097     error('!!! Data has different Y units !!!');
0098 end
0099
0100 a1_length = length(a1.getX);
0101 a2_length = length(a2.getX);
0102 gaptime = a2.getX(1) - a1.getX(a1_length);
0103 gapn = gaptime*a1.fs-1;
0104 t = (0:1:gapn-1)/a1.fs;
0105
0106 %--- gapfilling process itself
0107 if strcmp(method, 'linear')
0108     % linear interpolation method ---xfilled=(deltay/deltax)*t+y1(length(y1))---
0109     if length(a1.getY)>10 && length(a2.getY)>10
0110         dy = mean(a2.getY(1:10))-mean(a1.getY(a1_length-10:a1_length));
0111
0112         filling_data = (dy/gaptime)*t + mean(a1.getY(a1_length-10:a1_length));
0113         filling_time = (1:1:gapn)/a1.fs + a1.getX(a1_length);
0114     else
0115         error('!!! Not enough data in the data segments (min=11 for each one for the
linear method) !!!');
0116     end
0117 elseif strcmp(method, 'spline') % spline method xfilled = a*T^3 + b*T^2 + c*T +d
0118
0119     if length(a1.getY)>1000 && length(a2.getY)>1000
0120
0121         % derivatives of the input data are calculated
0122         da1 = diff(a1.getY(1:100:a1_length))*(a1.fs/100);
0123         da1 = tsdata(da1, a1.fs/100);
0124         da1 = ao(da1);
0125
0126         da2 = diff(a2.getY(1:100:a2_length))*(a2.fs/100);
0127         da2 = tsdata(da2, a2.fs/100);
0128         da2 = ao(da2);
0129
0130
0131         % This filters the previous derivatives
0132         % filters parameters are obtained
0133         plfa1 = getlpFilter(a1.fs/100);
0134         plfa2 = getlpFilter(a2.fs/100);
0135
0136         lpfa1 = miir(plfa1);
0137         lpfpla1 = plist(param('filter', lpfa1));
0138
0139         lpfa2 = miir(plfa2);
0140         lpfpla2 = plist(param('filter', lpfa2));
0141
0142         % derivatives are low-pass filtered
0143         da1filtered = filtfilt(da1, lpfpla1);
0144         da2filtered = filtfilt(da2, lpfpla2);
0145
0146         % coefficients are calculated
0147         c = mean(da1filtered.data.getY(length(da1filtered.data.getX)...
0148             -10:length(da1filtered.data.getX)));
0149         d = mean(a1.getY(length(a1.getY)-10:length(a1.getY)));
0150
0151         a=(2*d+(c+mean(da2filtered.data.getY(1:10)))...
0152             *gaptime-2*mean(a2.getY(1:10)))/(gaptime.^3);
0153
0154         b=- (3*d+2*c*gaptime+mean(da2filtered.data.getY(1:10)))...
0155             *gaptime-3*mean(a2.getY(1:10))/(gaptime^2);
0156
0157         % filling data is calculated with the coefficients a, b, c and d

```

```

0158     filling_data = a*t.^3+b*t.^2+c*t+d;
0159     filling_time = (1:1:gapn)/a1.fs + a1.getX(a1_length);
0160 else
0161     error('!!! Not enough data in data segments (min=1001 in spline method)');
0162 end
0163
0164 end
0165
0166 % this add noise (if desired) to the filled gap
0167 if strcmp(addnoise,'yes');
0168     % calculation of the standard deviation after eliminating the low-frequency
component
0169     phpf = gethpFilter(a1.fs);
0170     ax = tsdata(a1.getY, a1.fs);
0171     ax = ao(ax);
0172     hpf = miir(phpf);
0173     hpfpl = plist(param('filter', hpf));
0174     xhpf = filter(ax, hpfpl);
0175     hfnoise = std(xhpf);
0176
0177     % noise is added to the filling data
0178     filling_data = filling_data + randn(length(filling_data),1)*hfnoise.data.getY;
0179 end
0180
0181 % join data
0182 filling_data = [a1.getY; filling_data; a2.getY];
0183 filling_time = [a1.getX; filling_time; a2.getX];
0184
0185 % create new output tsdata
0186 ts = tsdata(filling_time, filling_data);
0187 ts.setYunits(a1.yunits);
0188 ts.setXunits(a1.xunits);
0189
0190 % create new output history
0191 %%histin = [as(1).hist as(2).hist];
0192 %%h = history(ALGONAME, VERSION, pls, histin);
0193 %%h = set(h, 'invars', invars);
0194
0195 % make output analysis object
0196 b = ao(ts);
0197 b.setName(sprintf('gapfilling(%s&%s)', ao_invars{1}, ao_invars{2}), 'internal');
0198 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%%');
0199 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%%');
0200 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%%');
0201 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%%');
0202 b.addHistory(getInfo(), pls, [ao_invars(1) ao_invars(2)], [as(1).hist as(2).hist]);
0203
0204 varargout{1} = b;
0205
0206 end
0207
0208 function plf = getlpFilter(x)
0209
0210     plf = plist();
0211     plf = append(plf, param('gain', 1));
0212     plf = append(plf, param('ripple', 0.5));
0213     plf = append(plf, param('type', 'lowpass'));
0214     plf = append(plf, param('order', 2));
0215     plf = append(plf, param('fs', x));
0216     plf = append(plf, param('fc', 0.1/100));
0217
0218 end
0219
0220 function phf = gethpFilter(x)
0221
0222     phf = plist();
0223     phf = append(phf, param('gain', 1));
0224     phf = append(phf, param('ripple', 0.5));
0225     phf = append(phf, param('type', 'highpass'));
0226     phf = append(phf, param('order', 2));
0227     phf = append(phf, param('fs', x));
0228     phf = append(phf, param('fc', 0.1/100));
0229
0230 end
0231
0232 %%%%%%%%%%%
0233 %
0234 %%%%%%%%%%% Local Functions %%%%%%%%%%%
0235
0236 %%%%%%%%%%%
0237 %
0238 % FUNCTION:      getInfo
0239 %
0240 % DESCRIPTION:  Get Info Object
0241 %
0242 % HISTORY:      11-07-07 M Hewitson

```

```
0243 %                               Creation.
0244 %
0245 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0246
0247 function ii = getInfo(varargin)
0248
0249     if nargin == 1 && strcmpi(varargin{1}, 'None')
0250         sets = {};
0251         pl    = [];
0252     else
0253         sets = {'Default'};
0254         pl    = getDefaultPlist();
0255     end
0256     % Build info object
0257     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: gapfilling.m,v
1.5 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0258
0259 end
0260
0261 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0262 %
0263 % FUNCTION:      getDefaultPlist
0264 %
0265 % DESCRIPTION:  Get Default Plist
0266 %
0267 % HISTORY:      11-07-07 M Hewitson
0268 %               Creation.
0269 %
0270 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0271
0272 function plo = getDefaultPlist()
0273
0274     plo = plist();
0275     plo = append(plo, param('method', 'linear'));
0276     plo = append(plo, param('addnoise', 'no'));
0277
0278 end
0279
```

hist

PURPOSE ^

HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.

SYNOPSIS ^

function varargout = hist(varargin)

DESCRIPTION ^

```
HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: HIST overloads the histogram function (hist) of MATLAB for
Analysis Objects.

CALL:      b = hist(a)
           b = hist(a, pl)

INPUTS:    a - input analysis object(s)
           pl - a parameter list

OUTPUTS:    b - xydata type analysis object(s) containing the
               histogrammed data

PARAMETERS: 'N'      - number of bins [default: 10]
           'X'      - set of bin centers

WARNING: the '.' method of calling hist() doesn't work since AOs have a
property called 'hist'. Use the standard function call instead:

    >> a.hist % returns the history object and doesn't call ao/hist
    >> hist(a) % calls ao/hist

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('hist')

               Get information about a specified set-plist by calling:
    >> ao.getInfo('hist', 'Number of bins')

VERSION:    $Id: hist.m,v 1.23 2008/09/05 11:05:29 ingo Exp $

HISTORY:    24-05-07 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [hist](#) HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
-  [x](#) X Get the data property 'x'.

This function is called by:

-  [hist](#) HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.

SUBFUNCTIONS ^

 function ii = getInfo(varargin)
 function plo = getDefaultPlist(set)

SOURCE CODE ^

```

0001 % HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: HIST overloads the histogram function (hist) of MATLAB for
0005 %               Analysis Objects.
0006 %
0007 % CALL:         b = hist(a)
0008 %               b = hist(a, pl)
0009 %
0010 % INPUTS:       a - input analysis object(s)
0011 %               pl - a parameter list
0012 %
0013 % OUTPUTS:      b - xydata type analysis object(s) containing the
0014 %               histogrammed data
0015 %
0016 % PARAMETERS:   'N' - number of bins [default: 10]
0017 %               'X' - set of bin centers
0018 %
0019 % WARNING: the '.' method of calling hist() doesn't work since AOs have a
0020 % property called 'hist'. Use the standard function call instead:
0021 %
0022 %     >> a.hist % returns the history object and doesn't call ao/hist
0023 %     >> hist(a) % calls ao/hist
0024 %
0025 % M-FILE INFO: Get information about this methods by calling
0026 %               >> ao.getInfo('hist')
0027 %
0028 %               Get information about a specified set-plist by calling:
0029 %               >> ao.getInfo('hist', 'Number of bins')
0030 %
0031 % VERSION:      $Id: hist.m,v 1.23 2008/09/05 11:05:29 ingo Exp $
0032 %
0033 % HISTORY:      24-05-07 M Hewitson
0034 %               Creation
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037
0038 function varargout = hist(varargin)
0039
0040 % Check if this is a call for parameters
0041 if utils.helper.isinfocall(varargin{:})
0042     varargout{1} = getInfo(varargin{3});
0043     return
0044 end
0045
0046 import utils.const.*
0047 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0048
0049 % Collect input variable names
0050 in_names = cell(size(varargin));
0051 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0052
0053 % Collect all AOs and plists
0054 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0055 [pli, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0056
0057 pl = combine(pli, getDefaultPlist('Number of bins'));
0058
0059 % Decide on a deep copy or a modify
0060 bs = copy(as, nargout);
0061
0062 % Get parameters
0063 N = find(pl, 'N');
0064 X = find(pl, 'X');
0065
0066 %----- Loop over input AOs
0067
0068 % start looping
0069 for j=1: numel(bs)
0070     % Histogram this data
0071     if isempty(X)
0072         [n,x] = hist(bs(j).data.y, N);
0073     else
0074         [n,x] = hist(bs(j).data.y, X);
0075     end
0076     % Keep the data shape of the input AO
0077     if size(bs(j).data.y, 1) ~= 1

```

```

0078         x = x.';
0079         n = n.';
0080     end
0081     % make a new xydata object
0082     xy = xydata(x, n);
0083     xy.setXunits(bs(j).data.yunits);
0084     xy.setYunits('Count');
0085     % make output analysis object
0086     bs(j).data = xy;
0087     % name for this object
0088     bs(j).setName(sprintf('hist(%s)', ao_invars{j}), 'internal');
0089     % Add history
0090     bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0091     % Add to outputs
0092 end % end of AO loop
0093
0094 % Set outputs
0095 varargout{1} = bs;
0096 end
0097
0098 %-----
0099 % Get Info Object
0100 %-----
0101 function ii = getInfo(varargin)
0102     if nargin == 1 && strcmpi(varargin{1}, 'None')
0103         sets = {};
0104         pls = [];
0105     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0106         sets{1} = varargin{1};
0107         pls = getDefaultPlist(sets{1});
0108     else
0109         sets = {'Number of bins', 'Bin centres'};
0110         pls = [];
0111         for kk=1:numel(sets)
0112             pls = [pls getDefaultPlist(sets{kk})];
0113         end
0114     end
0115     % Build info object
0116     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: hist.m,v 1.23
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0117 end
0118
0119 %-----
0120 % Get Default Plist
0121 %-----
0122
0123 function plo = getDefaultPlist(set)
0124     switch set
0125     case 'Number of bins'
0126         plo = plist('N', 10);
0127     case 'Bin centres'
0128         plo = plist('X', []);
0129     otherwise
0130         error('### Unknown default plist for the set [%s]', set);
0131     end
0132 end
0133
0134

```


ifft

PURPOSE ^

IFFT overloads the ifft operator for Analysis objects.

SYNOPSIS ^

function varargout = ifft(varargin)

DESCRIPTION ^

```
IFFT overloads the ifft operator for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: IFFT overloads the ifft operator for Analysis objects.

CALL:      b = ifft(a, pl)

PARAMETERS:
    'type'   - symmetric or nonsymmetric [default: 'symmetric']
    'twoside' - generate a two-sided spectrum from the single
                  sided input. Specify 'yes' or 'no' [default: yes].

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('ifft')

              Get information about a specified set-plist by calling:
    >> ao.getInfo('ifft', 'None')

VERSION:     $Id: ifft.m,v 1.11 2008/09/08 08:30:56 hewitson Exp $

HISTORY:     26-05-08 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [ifft](#) IFFT overloads the ifft operator for Analysis objects.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input PSD.
-  [ifft](#) IFFT overloads the ifft operator for Analysis objects.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl_default = getDefaultPlist()

SOURCE CODE ^


```

0001 % IFFT overloads the ifft operator for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: IFFT overloads the ifft operator for Analysis objects.
0005 %
0006 % CALL:          b = ifft(a, pl)
0007 %
0008 % PARAMETERS:
0009 %             'type'    - symmetric or nonsymmetric [default: 'symmetric']
0010 %             'twoside' - generate a two-sided spectrum from the single
0011 %                         sided input. Specify 'yes' or 'no' [default: yes].
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> ao.getInfo('ifft')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> ao.getInfo('ifft', 'None')
0018 %
0019 % VERSION:      $Id: ifft.m,v 1.11 2008/09/08 08:30:56 hewitson Exp $
0020 %
0021 % HISTORY:      26-05-08 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = ifft(varargin)
0027
0028 % Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 import utils.const.*
0035 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0036
0037 % Collect input variable names
0038 in_names = cell(size(varargin));
0039 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0040
0041 % Collect all AOs and plists
0042 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0043 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0044
0045 % Decide on a deep copy or a modify
0046 bs = copy(as, nargsout);
0047
0048 % combine plists
0049 pl = combine(pl, getDefaultPlist());
0050
0051 % two-sided or one?
0052 type    = find(pl, 'type');
0053 twoside = find(pl, 'twoside');
0054
0055 % Check input analysis object
0056 for j=1:numel(bs)
0057     switch class(bs(j).data)
0058     case {'fsdata'}
0059         if strcmpi(twoside, 'YES')
0060             y = [bs(j).data.getY; bs(j).data.getY(end-1:-1:2)];
0061         else
0062             y = bs(j).data.y;
0063         end
0064         % make a new fsdata object
0065         y = ifft(y, type);
0066         % compute time axis
0067         N = length(y);
0068         x = linspace(0, (N-1)/bs(j).data.fs, N);
0069         % Keep the data shape if the input AO
0070         if size(bs(j).data.y,1) == 1
0071             x = x.';
0072             y = y.';
0073         end
0074         fsd = tsdata(x, y);
0075         fsd.setXunits('s');
0076         fsd.setYunits('arb');
0077
0078         % Set data
0079         bs(j).data = fsd;
0080         % Set name
0081         bs(j).setName(sprintf('ifft(%s)', ao_invars{j}), 'internal');
0082         % Add history
0083         bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0084
0085     case {'tsdata', 'cdata', 'xydata'}
0086         error('### I don''t work for time-series, constant or x/y data.');
```

```

0087         otherwise
0088             error('### unknown data type.')
0089         end
0090     end
0091
0092     % Set output
0093     if nargout > 0
0094         varargout{1} = bs;
0095     end
0096 end
0097
0098 %-----
0099 % Get Info Object
0100 %-----
0101 function ii = getInfo(varargin)
0102     if nargin == 1 && strcmpi(varargin{1}, 'None')
0103         sets = {};
0104         pl = [];
0105     else
0106         sets = {'Default'};
0107         pl = getDefaultPlist;
0108     end
0109     % Build info object
0110     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: ifft.m,v 1.11
2008/09/08 08:30:56 hewitson Exp $', sets, pl);
0111 end
0112
0113 %-----
0114 % Get Default Plist
0115 %-----
0116 function pl_default = getDefaultPlist()
0117     pl_default = plist('type', 'symmetric', 'twoside', 'yes');
0118 end
0119
0120

```

interpmmissing

PURPOSE ^

INTERPMISSING interpolate missing samples in a time-series.

SYNOPSIS ^

function varargout = interpmmissing(varargin)

DESCRIPTION ^

```
INTERPMISSING interpolate missing samples in a time-series.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

INTERPMISSING interpolate missing samples in a time-series. Missing samples
are identified as being those where the time-span between one
sample and the next is larger than d/fs where d is a
tolerance value. Missing data is then placed in the gap in
steps of 1/fs. Obviously this is only really correct for
evenly sampled time-series.

CALL:          bs = interpmmissing(as)

INPUTS:        as  - array of analysis objects
                pl  - parameter list (see below)

OUTPUTS:       bs  - array of analysis objects, one for each input

PARAMETER LIST:
    'd'        - specify tolerance for finding missing samples.
                  [default: 1.5]

M-FILE INFO:   Get information about this methods by calling
                >> ao.getInfo('interpmmissing')

                Get information about a specified set-plist by calling:
                >> ao.getInfo('interpmmissing', 'None')

VERSION:       $Id: interpmmissing.m,v 1.12 2008/09/08 08:31:18 hewitson Exp $

HISTORY:       22-05-08 M Hewitson
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [diff](#) DIFF differentiates the data in AO.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [interp](#) INTERP interpolate the values in the input AO(s) at new values.

This function is called by:

-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % INTERPMISSING interpolate missing samples in a time-series.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % INTERPMISSING interpolate missing samples in a time-series. Missing samples
0005 % are identified as being those where the time-span between one
0006 % sample and the next is larger than d/fs where d is a
0007 % tolerance value. Missing data is then placed in the gap in
0008 % steps of 1/fs. Obviously this is only really correct for
0009 % evenly sampled time-series.
0010 %
0011 % CALL:          bs = interpmisssing(as)
0012 %
0013 % INPUTS:        as - array of analysis objects
0014 %                pl - parameter list (see below)
0015 %
0016 % OUTPUTS:       bs - array of analysis objects, one for each input
0017 %
0018 % PARAMETER LIST:
0019 %                'd' - specify tolerance for finding missing samples.
0020 %                    [default: 1.5]
0021 %
0022 % M-FILE INFO:   Get information about this methods by calling
0023 %                >> ao.getInfo('interpmisssing')
0024 %
0025 %                Get information about a specified set-plist by calling:
0026 %                >> ao.getInfo('interpmisssing', 'None')
0027 %
0028 % VERSION:       $Id: interpmisssing.m,v 1.12 2008/09/08 08:31:18 hewitson Exp $
0029 %
0030 % HISTORY:       22-05-08 M Hewitson
0031 %                Creation
0032 %
0033 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034
0035 function varargout = interpmisssing(varargin)
0036
0037 % Check if this is a call for parameters
0038 if utils.helper.isinfocall(varargin{:})
0039     varargout{1} = getInfo(varargin{3});
0040     return
0041 end
0042
0043 import utils.const.*
0044 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0045
0046 % Collect input variable names
0047 in_names = cell(size(varargin));
0048 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0049
0050 % Collect all AOs and plists
0051 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0052 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0053
0054 % Decide on a deep copy or a modify
0055 bs = copy(as, nargout);
0056
0057 % Combine plists
0058 pl = combine(pl, getDefaultPlist);
0059
0060 % Get tolerance
0061 dtol = find(pl, 'd');
0062
0063 % Get only tsdata AOs
0064 for j=1: numel(bs)
0065     if isa(bs(j).data, 'tsdata')
0066
0067         % capture input history
0068         ih = bs(j).hist;
0069
0070         % find missing samples
0071         t = [];
0072         d = diff(bs(j).data.getX);
0073         idxs = find(d>dtol/bs(j).data.fs);
0074         utils.helper.msg(msg.PROC1, 'found %d data gaps', numel(idxs));
0075
0076         % create new time grid
0077         count = 0;
0078         for k=1: numel(idxs)
0079             idx = idxs(k);
0080             if isempty(t)
0081                 t = bs(j).data.getX(1+idx-idxs(1):idx);

```

```

0082     end
0083     % now add samples at 1/fs until we are within 1/fs of the next sample
0084     while bs(j).data.getX(idx+1)-t(end) > 1/bs(j).data.fs
0085         t(end+1) = t(end)+1/bs(j).data.fs;
0086         count = count + 1;
0087     end
0088     if k==numel(idxs)
0089         t = [t; bs(j).data.getX(idx+1:end)];
0090     else
0091         t = [t; bs(j).data.getX(idx+1:idxs(k+1))];
0092     end
0093 end
0094 utils.helper.msg(msg.PROC1, 'filled with %d samples', count);
0095
0096 % now interpolate onto this new time-grid
0097 if ~isempty(t)
0098     interp(bs(j), plist('vertices', t));
0099     bs(j).setName(sprintf('interpmmissing(%s)', ao_invars{j}), 'internal');
0100     % Add history
0101     bs(j).addHistory(getInfo, pl, ao_invars(j), ih);
0102 else
0103     utils.helper.msg(msg.PROC1, 'no missing samples found in %s - no action
performed.', ao_invars{j});
0104 end
0105 else
0106     utils.helper.msg(msg.PROC1, 'skipping AO %s - it''s not a time-series AO.',
ao_invars{j});
0107 end
0108 end
0109
0110 % Set output
0111 if nargout > 0
0112     varargout{1} = bs;
0113 end
0114 end
0115
0116 %-----
0117 % Get Info Object
0118 %-----
0119 function ii = getInfo(varargin)
0120     if nargin == 1 && strcmpi(varargin{1}, 'None')
0121         sets = {};
0122         pl = [];
0123     else
0124         sets = {'Default'};
0125         pl = getDefaultPlist;
0126     end
0127     % Build info object
0128     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id:
interpmmissing.m,v 1.12 2008/09/08 08:31:18 hewitson Exp $', sets, pl);
0129 end
0130
0131 %-----
0132 % Get Default Plist
0133 %-----
0134 function pl_default = getDefaultPlist()
0135     pl_default = plist('d', 1.5);
0136 end
0137
0138

```

lcohere

PURPOSE ^

LCOHERE implement coherence estimation computed on a log frequency axis.

SYNOPSIS ^

function varargout = lcohere(varargin)

DESCRIPTION ^

```
LCOHERE implement coherence estimation computed on a log frequency axis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LCOHERE implement coherence estimation computed on a
log frequency axis.

Usage:
>> tfs = lcohere(as)
>> tfs = lcohere(as, pl)

Inputs:
as - array of analysis objects
pl - parameter list (see below)

Outputs:
bs - array of analysis objects, one for each input

Parameter list:
Kdes - desired number of averages [default: 100]
Lmin - minimum segment length [default: 0]
Jdes - number of spectral frequencies to compute [default: fs/4]
Win - a specwin window object
      Only the design parameters of the window object are used; the
      window is recomputed for each DFT length inside the ltpda_dft
      algorithm. [default: Kaiser with -200dB PSL]
Olap - desired overlap [default: taken from window]
Order - order of detrending.
        -1 - no detrending
         0 - subtract mean
         1 - subtract linear fit
         N - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('lcohere')

              Get information about a specified set-plist by calling:
>> ao.getInfo('lcohere', 'None')

VERSION: $Id: lcohere.m,v 1.7 2008/09/05 11:05:29 ingo Exp $

HISTORY: M Hewitson 02-02-07
         Created
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE [^]

```

0001 % LCOHERE implement coherence estimation computed on a log frequency axis.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % LCOHERE implement coherence estimation computed on a
0005 % log frequency axis.
0006 %
0007 % Usage:
0008 %     >> tfs = lcohere(as)
0009 %     >> tfs = lcohere(as, pl)
0010 %
0011 % Inputs:
0012 %     as - array of analysis objects
0013 %     pl - parameter list (see below)
0014 %
0015 % Outputs:
0016 %     bs - array of analysis objects, one for each input
0017 %
0018 % Parameter list:
0019 %     Kdes - desired number of averages [default: 100]
0020 %     Lmin - minimum segment length [default: 0]
0021 %     Jdes - number of spectral frequencies to compute [default: fs/4]
0022 %     Win - a specwin window object
0023 %           Only the design parameters of the window object are used; the
0024 %           window is recomputed for each DFT length inside the ltpda_dft
0025 %           algorithm. [default: Kaiser with -200dB PSL]
0026 %     Olap - desired overlap [default: taken from window]
0027 %     Order - order of detrending.
0028 %           -1 - no detrending
0029 %           0 - subtract mean
0030 %           1 - subtract linear fit
0031 %           N - subtract fit of polynomial, order N
0032 %
0033 % M-FILE INFO: Get information about this methods by calling
0034 %     >> ao.getInfo('lcohere')
0035 %
0036 %           Get information about a specified set-plist by calling:
0037 %     >> ao.getInfo('lcohere', 'None')
0038 %
0039 % VERSION: $Id: lcohere.m,v 1.7 2008/09/05 11:05:29 ingo Exp $
0040 %
0041 % HISTORY: M Hewitson 02-02-07
0042 %           Created
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %
0046 function varargout = lcohere(varargin)
0047 %
0048 % Check if this is a call for parameters
0049 if utils.helper.isinfocall(varargin{:})
0050     varargout{1} = getInfo(varargin{3});
0051     return
0052 end
0053
0054 import utils.const.*
0055 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0056
0057 % Collect input variable names
0058 in_names = cell(size(varargin));
0059 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0060
0061 % Collect all AOs and plists
0062 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0063 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0064
0065 % Combine plists
0066 pl = combine(pl, getDefaultPlist);
0067
0068 % Compute coherence with lxspect
0069 varargout{1} = ao.lxspect(as, pl, 'cohere', getInfo, ao_invars);
0070
0071 end
0072
0073 %-----
0074 % Get Info Object
0075 %-----
0076 function ii = getInfo(varargin)
0077 if nargin == 1 && strcmpi(varargin{1}, 'None')
0078     sets = {};
0079     pl = [];
0080 else
0081     sets = {'Default'};

```

```
0082     pl     = getDefaultPlist;
0083     end
0084     % Build info object
0085     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: lcohere.m,v 1.7
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0086     end
0087
0088     %-----
0089     % Get Default Plist
0090     %-----
0091     function pl = getDefaultPlist()
0092         pl = plist('Kdes', 100, 'Jdes', 1000, 'Lmin', 0,...
0093             'win', getappdata(0, 'ltpda_default_spectral_window'), ...
0094             'Olap', -1, 'Order', 0);
0095     end
0096
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

lcpsd

PURPOSE ^

LCPSD implement cross-power-spectral density estimation computed on a log frequency axis.

SYNOPSIS ^

function varargout = lcpsd(varargin)

DESCRIPTION ^

```
LCPSD implement cross-power-spectral density estimation computed on a log frequency axis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LCPSD implement cross-power-spectral density estimation computed on a
log frequency axis.

Usage:
>> tfs = lcpsd(as)
>> tfs = lcpsd(as, pl)

Inputs:
as - array of analysis objects
pl - parameter list (see below)

Outputs:
bs - array of analysis objects, one for each input

Parameter list:
Kdes - desired number of averages [default: 100]
Lmin - minimum segment length [default: 0]
Jdes - number of spectral frequencies to compute [default: fs/4]
Win - a specwin window object
      Only the design parameters of the window object are used; the
      window is recomputed for each DFT length inside the ltpda_dft
      algorithm. [default: Kaiser with -200dB PSL]
Olap - desired overlap [default: taken from window]
Order - order of detrending.
        -1 - no detrending
         0 - subtract mean
         1 - subtract linear fit
         N - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('lcpsd')

              Get information about a specified set-plist by calling:
>> ao.getInfo('lcpsd', 'None')

VERSION: $Id: lcpsd.m,v 1.7 2008/09/05 11:05:29 ingo Exp $

HISTORY: M Hewitson 02-02-07
         Created

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE [^]

```

0001 % LCPSD implement cross-power-spectral density estimation computed on a log frequency
axis.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % LCPSD implement cross-power-spectral density estimation computed on a
0005 % log frequency axis.
0006 %
0007 % Usage:
0008 %     >> tfs = lcpsd(as)
0009 %     >> tfs = lcpsd(as, pl)
0010 %
0011 % Inputs:
0012 %     as - array of analysis objects
0013 %     pl - parameter list (see below)
0014 %
0015 % Outputs:
0016 %     bs - array of analysis objects, one for each input
0017 %
0018 % Parameter list:
0019 %     Kdes - desired number of averages [default: 100]
0020 %     Lmin - minimum segment length [default: 0]
0021 %     Jdes - number of spectral frequencies to compute [default: fs/4]
0022 %     Win - a specwin window object
0023 %           Only the design parameters of the window object are used; the
0024 %           window is recomputed for each DFT length inside the ltpda_dft
0025 %           algorithm. [default: Kaiser with -200dB PSL]
0026 %     Olap - desired overlap [default: taken from window]
0027 %     Order - order of detrending.
0028 %             -1 - no detrending
0029 %             0 - subtract mean
0030 %             1 - subtract linear fit
0031 %             N - subtract fit of polynomial, order N
0032 %
0033 % M-FILE INFO: Get information about this methods by calling
0034 %             >> ao.getInfo('lcpsd')
0035 %
0036 %             Get information about a specified set-plist by calling:
0037 %             >> ao.getInfo('lcpsd', 'None')
0038 %
0039 % VERSION: $Id: lcpsd.m,v 1.7 2008/09/05 11:05:29 ingo Exp $
0040 %
0041 % HISTORY: M Hewitson 02-02-07
0042 %           Created
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %
0046 function varargout = lcpsd(varargin)
0047
0048 %%% Check if this is a call for parameters
0049 if nargin == 3 && ischar(varargin{2}) && ischar(varargin{3}) && strcmp(varargin{2},
'INFO')
0050     varargout{1} = getInfo(varargin{3});
0051     return
0052 end
0053
0054 import utils.const.*
0055 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0056
0057 % Collect input variable names
0058 in_names = cell(size(varargin));
0059 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0060
0061 % Collect all AOs and plists
0062 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0063 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0064
0065 % Combine plists
0066 pl = combine(pl, getDefaultPlist);
0067
0068 % Compute coherence with lxspect
0069 varargout{1} = ao.lxspect(as, pl, 'cpsd', getInfo, ao_invars);
0070 end
0071
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073 %
0074 % FUNCTION:     getInfo
0075 %
0076 % DESCRIPTION:  Get Info Object
0077 %
0078 % HISTORY:      11-07-07 M Hewitson
0079 %               Creation.

```

```
0080 %
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082
0083 function ii = getInfo(varargin)
0084     if nargin == 1 && strcmpi(varargin{1}, 'None')
0085         sets = {};
0086         pl = [];
0087     else
0088         sets = {'Default'};
0089         pl = getDefaultPlist();
0090     end
0091     % Build info object
0092     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: lcpsd.m,v 1.7
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0093 end
0094
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096 %
0097 % FUNCTION:      getDefaultPlist
0098 %
0099 % DESCRIPTION:   Get Default Plist
0100 %
0101 % HISTORY:       11-07-07 M Hewitson
0102 %                Creation.
0103 %
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105
0106 function plo = getDefaultPlist()
0107     plo = plist('Kdes', 100, 'Jdes', 1000, 'Lmin', 0,...
0108         'win', getappdata(0, 'ltpda_default_spectral_window'), ...
0109         'Olap', -1, 'Order', 0);
0110
0111 end
0112
0113
0114
0115
0116
```

lincom

PURPOSE ^

LINCOM

SYNOPSIS ^

function varargout = lincom(varargin)

DESCRIPTION ^

```
LINCOM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LINCOM

CALL:      b = lincom(a1,a2,a3,...,pl)
           b = lincom(a1,a2,a3,...,aN)

           If no parameter list is specified, one of the analysis objects
           should contain a cdata type with the coefficients inside.

INPUTS:    aN - a list of analysis objects of the same type
           pl - a parameter list

OUTPUTS:    b - output analysis object

PARAMETERS: 'coeffs' - vector of coefficients, same length as number of input AOs

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('lincom')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('lincom', 'None')

VERSION:    $Id: lincom.m,v 1.9 2008/09/05 11:05:29 ingo Exp $

HISTORY:    14-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

0001 % LINCOM

```

0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LINCOM
0005 %
0006 % CALL:      b = lincom(a1,a2,a3,...,pl)
0007 %           b = lincom(a1,a2,a3,...,aN)
0008 %
0009 %           If no parameter list is specified, one of the analysis objects
0010 %           should contain a cdata type with the coefficients inside.
0011 %
0012 % INPUTS:    aN - a list of analysis objects of the same type
0013 %           pl - a parameter list
0014 %
0015 % OUTPUTS:   b - output analysis object
0016 %
0017 % PARAMETERS: 'coeffs' - vector of coefficients, same length as number of input AOs
0018 %
0019 % M-FILE INFO: Get information about this methods by calling
0020 %             >> ao.getInfo('lincom')
0021 %
0022 %             Get information about a specified set-plist by calling:
0023 %             >> ao.getInfo('lincom', 'None')
0024 %
0025 % VERSION:   $Id: lincom.m,v 1.9 2008/09/05 11:05:29 ingo Exp $
0026 %
0027 % HISTORY:   14-02-07 M Hewitson
0028 %           Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031
0032 function varargout = lincom(varargin)
0033
0034 %%% Check if this is a call for parameters
0035 if utils.helper.isinfocall(varargin{:})
0036     varargout{1} = getInfo(varargin{3});
0037     return
0038 end
0039
0040 import utils.const.*
0041 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0042
0043 %%% Collect input variable names
0044 in_names = cell(size(varargin));
0045 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0046
0047 %%% Collect all AOs
0048 [as, ao_invars, rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0049 [pli, pl_invars, rest] = utils.helper.collect_objects(rest, 'plist', in_names);
0050
0051 if nargout == 0
0052     error('### lincom cannot be used as a modifier. Please give an output variable.');
```

```

0088 for kk = 2:numel(ts);
0089     new_units = new_units + ts(kk).data.yunits;
0090 end
0091
0092 % Make linear combination and collect name string
0093 nstr = '';
0094 y = zeros(size(ts(1).data.y));
0095 for j=1:nc
0096     nstr = [nstr num2str(coeffs(j)) '*' ao_invars{j} ' + '];
0097     y = y + coeffs(j).*ts(j).data.y;
0098 end
0099 nstr = deblank(nstr(1:end-2));
0100
0101 % create new ao
0102 b = ao;
0103 b.data = copy(ts(1).data,1);
0104 b.data.setY(y);
0105
0106 %%% Set Name
0107 b.setName(nstr, 'internal');
0108
0109 %%% Set new Units
0110 b.setYunits(new_units, 'internal');
0111
0112 %%% Add History
0113 b.addHistory(getInfo('None'), pl, ao_invars, [ts.hist chist]);
0114
0115 %%% Reshape the ouput to the same size of the input
0116 varargout{1} = b;
0117 end
0118
0119 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0120 %                                     Local Functions                                     %
0121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122
0123 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0124 %
0125 % FUNCTION:      getInfo
0126 %
0127 % DESCRIPTION:  Get Info Object
0128 %
0129 % HISTORY:      11-07-07 M Hewitson
0130 %               Creation.
0131 %
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133
0134 function ii = getInfo(varargin)
0135     if nargin == 1 && strcmpi(varargin{1}, 'None')
0136         sets = {};
0137         pl = [];
0138     else
0139         sets = {'Default'};
0140         pl = getDefaultPlist;
0141     end
0142     % Build info object
0143     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: lincom.m,v 1.9
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0144 end
0145
0146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0147 %
0148 % FUNCTION:      getDefaultPlist
0149 %
0150 % DESCRIPTION:  Get Default Plist
0151 %
0152 % HISTORY:      11-07-07 M Hewitson
0153 %               Creation.
0154 %
0155 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0156
0157 function plo = getDefaultPlist()
0158     plo = plist();
0159 end
0160
0161

```

linedetect

PURPOSE ^

LINEDETECT find spectral lines in the ao/fsdata objects.

SYNOPSIS ^

function varargout = linedetect(varargin)

DESCRIPTION ^

```
LINEDETECT find spectral lines in the ao/fsdata objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LINEDETECT find spectral lines in the ao/fsdata objects.

CALL:      b = linedetect(a, pl)

PARAMETERS: N      - max number of lines to return [default: 10]
             fsearch - frequeuncy search interval [default: all]
             thresh  - a threshold to test normalised amplitude spectrum against
                       [default: 2]
             bw      - bandwidth over which to compute median [default: 20 samples]
             hc      - percent of outliers to exclude from median estimation (0-1)
                       [default: 0.8]

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('linedetect')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('linedetect', 'None')

VERSION:     $Id: linedetect.m,v 1.3 2008/09/05 11:05:29 ingo Exp $

HISTORY:     14-05-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [min](#) MIN computes the minimum value of the data in the AO.
-  [smoother](#) SMOOTHER smooths a given series of data points using the specified method.
-  [sort](#) SORT the values in the AO.
-  [y](#) Y Get the data property 'y'.

This function is called by:

SUBFUNCTIONS ^

-  [function lines = findLines\(ay, f, nay, thresh, N, fsearch\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE [^]

```

0001 % LINEDetect find spectral lines in the ao/fsdata objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LINEDetect find spectral lines in the ao/fsdata objects.
0005 %
0006 % CALL:          b = linedetect(a, pl)
0007 %
0008 % PARAMETERS:    N          - max number of lines to return [default: 10]
0009 %                fsearch    - frequency search interval [default: all]
0010 %                thresh     - a threshold to test normalised amplitude spectrum against
0011 %                             [default: 2]
0012 %                bw         - bandwidth over which to compute median [default: 20 samples]
0013 %                hc         - percent of outliers to exclude from median estimation (0-1)
0014 %                             [default: 0.8]
0015 %
0016 % M-FILE INFO:    Get information about this methods by calling
0017 %                >> ao.getInfo('linedetect')
0018 %
0019 %                Get information about a specified set-plist by calling:
0020 %                >> ao.getInfo('linedetect', 'None')
0021 %
0022 % VERSION:        $Id: linedetect.m,v 1.3 2008/09/05 11:05:29 ingo Exp $
0023 %
0024 % HISTORY:        14-05-2007 M Hewitson
0025 %                  Creation
0026 %
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 function varargout = linedetect(varargin)
0031
0032 % Check if this is a call for parameters
0033 if utils.helper.isinfocall(varargin{:})
0034     varargout{1} = getInfo(varargin{3});
0035     return
0036 end
0037
0038 if nargin == 0
0039     error('### cat cannot be used as a modifier. Please give an output variable.');
```



```

0083     f = [lines(:).f];
0084     y = [lines(:).a];
0085
0086     % Keep the data shpare of the input A0
0087     if size(bs(j).data.y, 2) == 1
0088         f = f.';
0089         y = y.';
0090     end
0091
0092     % Make output data
0093     fs = fsdata(f, y, bs(j).data.fs);
0094
0095     else
0096         error('### I can only find lines in frequency-series A0s.');
```

```

0097     end
0098
0099     %----- Make output A0
0100
0101     % make output analysis object
0102     bs(j).data = fs;
0103
0104     bs(j).setName(sprintf('lines(%s)', ao_invars{1}));
0105     bs(j).addHistory(getInfo('None'), pl, ao_invars(j), bs(j).hist);
0106 end
0107
0108 % return output
0109 varargout{1} = bs;
0110 end
0111
0112 %-----
0113 % find spectral lines
0114 function lines = findLines(ay, f, nay, thresh, N, fsearch)
0115
0116     % look for spectral lines
0117     l = 0;
0118     pmax = 0;
0119     pmaxi = 0;
0120     line = [];
0121     idx = find( f>=fsearch(1) & f<=fsearch(2) );
0122     for j=1:length(idx)
0123         v = nay(idx(j));
0124         if v > thresh
0125             if v > pmax
0126                 pmax = v;
0127                 pmaxi = idx(j);
0128             end
0129         else
0130             if pmax > 0
0131                 % find index when we have pmax
0132                 fidx = pmaxi; %(find(nay(1:idx(j)))==pmax));
0133                 l = l+1;
0134                 line(l).idx = fidx;
0135                 line(l).f = f(fidx);
0136                 line(l).a = ay(fidx);
0137             end
0138             pmax = 0;
0139         end
0140     end
0141
0142     % Select largest peaks
0143     lines = [];
0144     if ~isempty(line)
0145         [bl, lidx] = sort([line.a], 'descend');
0146         lidxs = lidx(1:min([N length(lidx)]));
0147         lines = line(lidxs);
0148         disp(sprintf(' + found %d lines.', length([lines.f])));
0149     else
0150         disp(' + found 0 lines.');
```

```

0151     end
0152 end
0153
0154 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0155 %
0156 % Local Functions
0157 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0158 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0159 %
0160 % FUNCTION:      getInfo
0161 %
0162 % DESCRIPTION:  Get Info Object
0163 %
0164 % HISTORY:      11-07-07 M Hewitson
0165 %               Creation.
0166 %
0167 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0168
```

```
0169 function ii = getInfo(varargin)
0170     if nargin == 1 && strcmpi(varargin{1}, 'None')
0171         sets = {};
0172         pl = [];
0173     else
0174         sets = {'Default'};
0175         pl = getDefaultPlist;
0176     end
0177     % Build info object
0178     ii = minfo(mfilename, 'm', '', utils.const.categories.sigproc, '$Id: linedetect.m,v
1.3 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0179 end
0180
0181 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0182 %
0183 % FUNCTION:      getDefaultPlist
0184 %
0185 % DESCRIPTION:  Get Default Plist
0186 %
0187 % HISTORY:      11-07-07 M Hewitson
0188 %                Creation.
0189 %
0190 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0191
0192 function plo = getDefaultPlist()
0193     plo = plist('N', 10, 'fsearch', [0 1e10], 'thresh', 2, 'bw', 20, 'hc', 0.8);
0194 end
0195
```

lpsd

PURPOSE ^

LPSD implement the LPSD algorithm for analysis objects.

SYNOPSIS ^

function varargout = lpsd(varargin)

DESCRIPTION ^

```
LPSD implement the LPSD algorithm for analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LPSD implement the LPSD algorithm for analysis objects.

CALL:      bs = lpsd(as)

INPUTS:    as - array of analysis objects
           pl - parameter list (see below)

OUTPUTS:    bs - array of analysis objects, one for each input

PARAMETER LIST:

    Kdes - desired number of averages (default 100)
    Lmin - minimum segment length (default 0)
    Jdes - number of spectral frequencies to compute (default fs/2)
    win  - a specwin window object
           Only the design parameters of the window object are used; the
           window is recomputed for each DFT length inside the lpsd_core
           algorithm.
    Olap - desired overlap percentage (default: taken from window)
    Order - order of detrending
           -1 - no detrending
           0 - subtract mean
           1 - subtract linear fit
           N - subtract fit of polynomial, order N
    Scale - Scaling of output. Choose from:
           AS - Amplitude (linear) Spectrum
           ASD - Amplitude (linear) Spectral Density
           PS - Power Spectrum
           PSD - Power Spectral Density [default]

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('lpsd')

           Get information about a specified set-plist by calling:
>> ao.getInfo('lpsd', 'None')

VERSION:    $Id: lpsd.m,v 1.13 2008/09/05 14:14:58 hewitson Exp $

HISTORY:    02-02-07 M Hewitson
           Created

References:  "Improved spectrum estimation from digitized time series
           on a logarithmic frequency axis", Michael Troebs, Gerhard Heinzel,
           Measurement 39 (2006) 120-129.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [sqrt](#) SQRT computes the square root of the data in the AO.

This function is called by:

SUBFUNCTIONS [^]

 function ii = getInfo(varargin)

 function pl = getDefaultPlist()

SOURCE CODE [^]

```

0001 % LPSD implement the LPSD algorithm for analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LPSD implement the LPSD algorithm for analysis objects.
0005 %
0006 % CALL:          bs = lpsd(as)
0007 %
0008 % INPUTS:        as - array of analysis objects
0009 %                pl - parameter list (see below)
0010 %
0011 % OUTPUTS:       bs - array of analysis objects, one for each input
0012 %
0013 % PARAMETER LIST:
0014 %
0015 %     Kdes - desired number of averages (default 100)
0016 %     Lmin - minimum segment length (default 0)
0017 %     Jdes - number of spectral frequencies to compute (default fs/2)
0018 %     win  - a specwin window object
0019 %           Only the design parameters of the window object are used; the
0020 %           window is recomputed for each DFT length inside the lpsd_core
0021 %           algorithm.
0022 %     Olap - desired overlap percentage (default: taken from window)
0023 %     Order - order of detrending
0024 %            -1 - no detrending
0025 %            0 - subtract mean
0026 %            1 - subtract linear fit
0027 %            N - subtract fit of polynomial, order N
0028 %     Scale - Scaling of output. Choose from:
0029 %            AS - Amplitude (linear) Spectrum
0030 %            ASD - Amplitude (linear) Spectral Density
0031 %            PS - Power Spectrum
0032 %            PSD - Power Spectral Density [default]
0033 %
0034 % M-FILE INFO: Get information about this methods by calling
0035 %              >> ao.getInfo('lpsd')
0036 %
0037 %              Get information about a specified set-plist by calling:
0038 %              >> ao.getInfo('lpsd', 'None')
0039 %
0040 % VERSION:      $Id: lpsd.m,v 1.13 2008/09/05 14:14:58 hewitson Exp $
0041 %
0042 % HISTORY:      02-02-07 M Hewitson
0043 %                Created
0044 %
0045 % References:    "Improved spectrum estimation from digitized time series
0046 %                on a logarithmic frequency axis", Michael Troebs, Gerhard Heinzl,
0047 %                Measurement 39 (2006) 120-129.
0048 %
0049 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0050
0051 function varargout = lpsd(varargin)
0052
0053     import utils.const.*
0054
0055     % Check if this is a call for parameters
0056     if utils.helper.isinfocall(varargin{:})
0057         varargout{1} = getInfo(varargin{3});
0058         return
0059     end
0060
0061     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0062
0063     % Collect input variable names
0064     in_names = cell(size(varargin));
0065     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0066
0067     % Collect all AOs and plists
0068     [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0069     pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0070

```

```

0071 % Decide on a deep copy or a modify
0072 bs = copy(as, nargout);
0073
0074 % Combine plists
0075 pl = combine(pl, getDefaultPlist);
0076
0077 % Go through each input A0
0078 for j=1:numel(bs)
0079 % check this is a time-series object
0080 if ~isa(bs(j).data, 'tsdata')
0081 warning('!!! lpsd requires tsdata (time-series) inputs. Skipping A0 %s',
ao_invars{j});
0082 else
0083 % Desired number of averages
0084 Kdes = find(pl, 'Kdes');
0085 % num desired spectral frequencies
0086 Jdes = find(pl, 'Jdes');
0087 % Window function
0088 Win = find(pl, 'Win');
0089 % Overlap
0090 Nlap = find(pl, 'Olap');
0091 if Nlap < 0
0092 % use window function rov
0093 Nlap = Win.rov/100;
0094 utils.helper.msg(msg.PROC1, 'using recommended overlap of %d', Nlap);
0095 pl.pset('Olap', Nlap);
0096 end
0097 % Order of detrending
0098 Order = find(pl, 'Order');
0099 % Minimum segment length
0100 Lmin = find(pl, 'Lmin');
0101 % Get frequency vector
0102 [f, r, m, L, K] = ao.ltf_plan(length(bs(j).data.y), bs(j).data.fs, Nlap, 1, Lmin,
Jdes, Kdes);
0103 % compute LPSD
0104 try
0105 if find(pl, 'M-FILE ONLY')
0106 % Using pure m-file version
0107 [P, Pxx, ENBW] = ao.mlpsd_m(bs(j).data.y, f, r, m, L, bs(j).data.fs, Win,
Order, Nlap);
0108 else
0109 [P, Pxx, ENBW] = ao.mlpsd_mex(bs(j).data.y, f, r, m, L, bs(j).data.fs, Win,
Order, Win.rov, Lmin);
0110 end
0111 catch
0112 warning('!!! mex file dft failed. Using m-file version of lpsd.');
```

```

0113 % Using pure m-file version
0114 [P, Pxx, ENBW] = ao.mlpsd_m(bs(j).data.y, f, r, m, L, bs(j).data.fs, Win,
Order, Nlap);
0115 end
0116
0117 % Keep the data shape of the input A0
0118 if size(bs(j).data.y,1) == 1
0119 P = P.';
0120 Pxx = Pxx.';
0121 f = f.';
0122 end
0123
0124 % create new output fsdata
0125 scale = find(pl, 'Scale');
0126 switch scale
0127 case 'AS'
0128 fsd = fsdata(f, sqrt(P), bs(j).data.fs);
0129 fsd.setYunits(bs(j).data.yunits);
0130 case 'ASD'
0131 fsd = fsdata(f, sqrt(Pxx), bs(j).data.fs);
0132 fsd.setYunits(bs(j).data.yunits / unit('Hz^0.5'));
0133 case 'PS'
0134 fsd = fsdata(f, P, bs(j).data.fs);
0135 fsd.setYunits(bs(j).data.yunits.^2);
0136 case 'PSD'
0137 fsd = fsdata(f, Pxx, bs(j).data.fs);
0138 fsd.setYunits(bs(j).data.yunits.^2/unit('Hz'));
0139 otherwise
0140 error(['### Unknown scaling:' scale]);
0141 end
0142 fsd.setXunits('Hz');
0143 fsd.setEnbw(ENBW);
0144 % make output analysis object
0145 bs(j).data = fsd;
0146 % set name
0147 bs(j).setName(sprintf('%s(%s)', scale, ao_invars{j}), 'internal');
0148 % Add history
0149 bs(j).addHistory(getInfo, pl, ao_invars{j}, bs(j).hist);
0150 % Add processing info
0151 bs(j).procinfo = plist('r', r, 'm', m, 'l', L, 'k', K);

```

```

0152
0153     end % End tsdata if/else
0154 end % End AO loop
0155
0156 % Set output
0157 varargin{1} = bs;
0158
0159 end
0160
0161 %-----
0162 % Get Info Object
0163 %-----
0164 function ii = getInfo(varargin)
0165     if nargin == 1 && strcmpi(varargin{1}, 'None')
0166         sets = {};
0167         pl = [];
0168     else
0169         sets = {'Default'};
0170         pl = getDefaultPlist;
0171     end
0172     % Build info object
0173     ii = minfo(mfilename, 'ao', '', 'Signal Processing', '$Id: lpsd.m,v 1.13 2008/09/05
14:14:58 hewitson Exp $', sets, pl);
0174 end
0175
0176 %-----
0177 % Get Default Plist
0178 %-----
0179 function pl = getDefaultPlist()
0180     pl = plist('Kdes', 100, 'Jdes', 1000, 'Lmin', 0, ...
0181         'win', getappdata(0, 'ltpda_default_spectral_window'), ...
0182         'Olap', -1, 'Order', 0, 'Scale', 'PSD');
0183 end
0184

```

ltfe

PURPOSE ^

LTFE implement transfer-function estimation computed on a log frequency axis.

SYNOPSIS ^

function varargout = ltfe(varargin)

DESCRIPTION ^

```
LTFE implement transfer-function estimation computed on a log frequency axis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTFE implement transfer-function estimation computed on a
log frequency axis.

CALL:      tfs = ltfe(as)
           tfs = ltfe(as, pl)

INPUTS:    as - array of analysis objects
           pl - parameter list (see below)

OUTPUTS:    bs - array of analysis objects, one for each input

PARAMETER LIST:

           Kdes - desired number of averages [default: 100]
           Lmin - minimum segment length [default: 0]
           Jdes - number of spectral frequencies to compute [default: fs/4]
           Win  - a specwin window object
                  Only the design parameters of the window object are used; the
                  window is recomputed for each DFT length inside the ltpda_dft
                  algorithm. [default: Kaiser with -200dB PSLL]
           Olap - desired overlap [default: taken from window]
           Order - order of detrending.
                  -1 - no detrending
                   0 - subtract mean
                   1 - subtract linear fit
                   N - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('ltfe')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('ltfe', 'None')

VERSION:     $Id: ltfe.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:     02-02-2007 M Hewitson
              Created

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```

0001 % LTFE implement transfer-function estimation computed on a log frequency axis.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LTFE implement transfer-function estimation computed on a
0005 %               log frequency axis.
0006 %
0007 % CALL:         tfs = ltfe(as)
0008 %               tfs = ltfe(as, pl)
0009 %
0010 % INPUTS:       as - array of analysis objects
0011 %               pl - parameter list (see below)
0012 %
0013 % OUTPUTS:      bs - array of analysis objects, one for each input
0014 %
0015 % PARAMETER LIST:
0016 %
0017 %               Kdes - desired number of averages [default: 100]
0018 %               Lmin - minimum segment length [default: 0]
0019 %               Jdes - number of spectral frequencies to compute [default: fs/4]
0020 %               Win  - a specwin window object
0021 %                   Only the design parameters of the window object are used; the
0022 %                   window is recomputed for each DFT length inside the ltpda_dft
0023 %                   algorithm. [default: Kaiser with -200dB PSL]
0024 %               Olap - desired overlap [default: taken from window]
0025 %               Order - order of detrending.
0026 %                   -1 - no detrending
0027 %                   0 - subtract mean
0028 %                   1 - subtract linear fit
0029 %                   N - subtract fit of polynomial, order N
0030 %
0031 % M-FILE INFO: Get information about this methods by calling
0032 %               >> ao.getInfo('ltfe')
0033 %
0034 %               Get information about a specified set-plist by calling:
0035 %               >> ao.getInfo('ltfe', 'None')
0036 %
0037 % VERSION:      $Id: ltfe.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0038 %
0039 % HISTORY:      02-02-2007 M Hewitson
0040 %               Created
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043
0044 function varargout = ltfe(varargin)
0045
0046 % Check if this is a call for parameters
0047 if utils.helper.isinfocall(varargin{:})
0048     varargout{1} = getInfo(varargin{3});
0049     return
0050 end
0051
0052 import utils.const.*
0053 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0054
0055 % Collect input variable names
0056 in_names = cell(size(varargin));
0057 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0058
0059 % Collect all AOs and plists
0060 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0061 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0062
0063 % Combine plists
0064 pl = combine(pl, getDefaultPlist);
0065
0066 % Compute coherence with lxspect
0067 varargout{1} = ao.lxspect(as, pl, 'tfe', getInfo, ao_invars);
0068 end
0069
0070 %-----
0071 % Get Info Object
0072 %-----
0073 function ii = getInfo(varargin)
0074 if nargin == 1 && strcmpi(varargin{1}, 'None')
0075     sets = {};
0076     pl = [];
0077 else
0078     sets = {'Default'};
0079     pl = getDefaultPlist;
0080 end
0081 % Build info object
0082 ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: ltfe.m,v 1.6

```



```
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0083 end
0084
0085 %-----
0086 % Get Default Plist
0087 %-----
0088 function pl = getDefaultPlist()
0089     pl = plist(...
0090         'Kdes', 100,...
0091         'Jdes', 1000,...
0092         'Lmin', 0,...
0093         'Win', getappdata(0, 'ltfda_default_spectral_window'), ...
0094         'Olap', -1,...
0095         'Order', 0);
0096 end
0097 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

psd

PURPOSE ^

PSD makes power spectral density estimates of the time-series objects

SYNOPSIS ^

function varargout = psd(varargin)

DESCRIPTION ^

```
PSD makes power spectral density estimates of the time-series objects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PSD makes power spectral density estimates of the
             time-series objects in the input analysis objects.

CALL:       b = psd(a1,a2,a3,...,pl)

INPUTS:     b      - output analysis objects
             aN     - input analysis objects
             pl     - input parameter list

             Makes PSD estimates of each input AO using the Welch Overlap method.

             If the last input argument is a parameter list (plist) it is used.
             The following parameters are recognised.

PARAMETERS: 'Win'   - a specwin window object [default: Kaiser -200dB psll]
             'Olap' - segment percent overlap [default: taken from window function]
             'Nfft' - number of samples in each fft [default: length of input data]
                   A string value containing the variable 'fs' can
                   also be used, e.g., plist('Nfft', '2*fs')
             'Scale' - one of
                       'ASD' - amplitude spectral density
                       'PSD' - power spectral density [default]
                       'AS'  - amplitude spectrum
                       'PS'  - power spectrum
             'Order' - order of segment detrending
                       -1 - no detrending
                       0  - subtract mean [default]
                       1  - subtract linear fit
                       N  - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
             >> ao.getInfo('psd')

             Get information about a specified set-plist by calling:
             >> ao.getInfo('psd', 'None')

VERSION:    $Id: psd.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:    07-02-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.

This function is called by:

-  **firwhiten** FIRWHITEN whitens the input time-series by building an FIR whitening filter.
-  **pwelch** PWELCH makes power spectral density estimates of the time-series objects

SUBFUNCTIONS [^]

-  **function ii = getInfo(varargin)**
-  **function pl = getDefaultPlist()**

SOURCE CODE [^]

```

0001 % PSD makes power spectral density estimates of the time-series objects
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PSD makes power spectral density estimates of the
0005 %               time-series objects in the input analysis objects.
0006 %
0007 % CALL:         b = psd(a1,a2,a3,...,pl)
0008 %
0009 % INPUTS:       b      - output analysis objects
0010 %               aN     - input analysis objects
0011 %               pl     - input parameter list
0012 %
0013 %               Makes PSD estimates of each input AO using the Welch Overlap method.
0014 %
0015 %               If the last input argument is a parameter list (plist) it is used.
0016 %               The following parameters are recognised.
0017 %
0018 % PARAMETERS:   'Win'    - a specwin window object [default: Kaiser -200dB psll]
0019 %               'Olap'   - segment percent overlap [default: taken from window function]
0020 %               'Nfft'   - number of samples in each fft [default: length of input data]
0021 %                   A string value containing the variable 'fs' can
0022 %                   also be used, e.g., plist('Nfft', '2*fs')
0023 %               'Scale'  - one of
0024 %                           'ASD' - amplitude spectral density
0025 %                           'PSD' - power spectral density [default]
0026 %                           'AS'  - amplitude spectrum
0027 %                           'PS'  - power spectrum
0028 %               'Order'  - order of segment detrending
0029 %                           -1 - no detrending
0030 %                           0 - subtract mean [default]
0031 %                           1 - subtract linear fit
0032 %                           N - subtract fit of polynomial, order N
0033 %
0034 % M-FILE INFO:   Get information about this methods by calling
0035 %               >> ao.getInfo('psd')
0036 %
0037 %               Get information about a specified set-plist by calling:
0038 %               >> ao.getInfo('psd', 'None')
0039 %
0040 % VERSION:       $Id: psd.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0041 %
0042 % HISTORY:       07-02-2007 M Hewitson
0043 %               Creation
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046
0047 function varargout = psd(varargin)
0048
0049 % Check if this is a call for parameters
0050 if utils.helper.isinfocall(varargin{:})
0051     varargout{1} = getInfo(varargin{3});
0052     return
0053 end
0054
0055 import utils.const.*
0056 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0057
0058 % Collect input variable names
0059 in_names = cell(size(varargin));
0060 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0061
0062 % Collect all AOs and plists
0063 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0064 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0065
0066 % Decide on a deep copy or a modify
0067 bs = copy(as, nargsout);
0068
0069 % combine plists

```

```

0070 pl = combine(pl, getDefaultPlist());
0071
0072 % Loop over input AOs
0073 for j=1:numel(bs)
0074     % check input data
0075     if isa(bs(j).data, 'tsdata')
0076         fs = bs(j).data.fs;
0077         %----- Check all defaults are sensible
0078         % Check the number of points in FFT. If this is not set (<0) we set it
0079         % to be the length of the input data.
0080         Nfft = find(pl, 'Nfft');
0081         setWindow = 0;
0082         if ischar(Nfft)
0083             nNfft = floor(eval(Nfft));
0084             utils.helper.msg(msg.PROC1, 'setting Nfft to %s = %d', Nfft, nNfft);
0085             Nfft = nNfft;
0086         else
0087             if Nfft <= 0
0088                 Nfft = length(bs(j).data.y);
0089                 utils.helper.msg(msg.PROC1, 'using default Nfft of %g', Nfft);
0090                 setWindow = 1;
0091             end
0092         end
0093         pl.pset('Nfft', Nfft);
0094
0095         % Check the window function. If the length of the window doesn't match
0096         % NFFT then we resize it.
0097         Win = find(pl, 'Win'); % Window object
0098         if setWindow || length(Win.win)~=Nfft
0099             switch Win.type
0100                 case {'Kaiser', 'Flatop'}
0101                     Win = specwin(Win.type, Nfft, Win.psl1);
0102                 otherwise
0103                     Win = specwin(Win.type, Nfft);
0104             end
0105             utils.helper.msg(msg.PROC1, 'reset window to %s(%d)', strrep(Win.type, '_',
0106 '\_'), length(Win.win));
0107             pl.pset('Win', Win);
0108
0109             % Check the overlap. If this is not set, we take the overlap from that
0110             % recommended by the window function.
0111             Olap = find(pl, 'Olap');
0112             if Olap <= 0
0113                 Olap = Win.rov;
0114                 utils.helper.msg(msg.PROC1, 'using default overlap of %2.1f%%', Olap);
0115             end
0116             pl.pset('Olap', Olap);
0117
0118             % scaling of spectrum
0119             scale = find(pl, 'Scale');
0120
0121             % check parameters
0122             if length(Win.win) ~= Nfft
0123                 error('### Window length should be same as Nfft.');
```

```
0154 %-----
0155 % Get Info Object
0156 %-----
0157 function ii = getInfo(varargin)
0158     if nargin == 1 && strcmpi(varargin{1}, 'None')
0159         sets = {};
0160         pl = [];
0161     else
0162         sets = {'Default'};
0163         pl = getDefaultPlist;
0164     end
0165     % Build info object
0166     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: psd.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0167 end
0168
0169 %-----
0170 % Get Default Plist
0171 %-----
0172 function pl = getDefaultPlist()
0173     pl = plist(...
0174         'Nfft', -1, ...
0175         'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
0176         'Olap', -1, ...
0177         'Scale', 'PSD', ...
0178         'Order', 0);
0179 end
0180 % END
0181
```

pwelch

PURPOSE ^

PWELCH makes power spectral density estimates of the time-series objects

SYNOPSIS ^

function varargout = pwelch(varargin)

DESCRIPTION ^

```
PWELCH makes power spectral density estimates of the time-series objects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PWELCH makes power spectral density estimates of the
             time-series objects in the input analysis objects.

CALL:       b = pwelch(a1,a2,a3,...,pl)

INPUTS:     b      - output analysis objects
             aN     - input analysis objects
             pl     - input parameter list

             Makes PSD estimates using psd() of each input A0.

             If the last input argument is a parameter list (plist) it is used.
             The following parameters are recognised.

PARAMETERS: 'Win'   - a specwin window object [default: Kaiser -200dB psll]
             'Olap' - segment percent overlap [default: taken from window function]
             'Nfft' - number of samples in each fft [default: length of input data]
                   A string value containing the variable 'fs' can
                   also be used, e.g., plist('Nfft', '2*fs')
             'Scale' - one of
                        'ASD' - amplitude spectral density
                        'PSD' - power spectral density [default]
                        'AS'  - amplitude spectrum
                        'PS'  - power spectrum
             'Order' - order of segment detrending
                        -1 - no detrending
                        0 - subtract mean [default]
                        1 - subtract linear fit
                        N - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
             >> ao.getInfo('pwelch')

             Get information about a specified set-plist by calling:
             >> ao.getInfo('pwelch', 'None')

VERSION:     $Id: pwelch.m,v 1.14 2008/08/07 13:00:39 ingo Exp $

HISTORY:     07-02-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [psd](#) PSD makes power spectral density estimates of the time-series objects

This function is called by:

SOURCE CODE ^

```

0001 % PWELCH makes power spectral density estimates of the time-series objects
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PWELCH makes power spectral density estimates of the
0005 %               time-series objects in the input analysis objects.
0006 %
0007 % CALL:         b = pwelch(a1,a2,a3,...,pl)
0008 %
0009 % INPUTS:       b      - output analysis objects
0010 %               aN     - input analysis objects
0011 %               pl     - input parameter list
0012 %
0013 %               Makes PSD estimates using psd() of each input A0.
0014 %
0015 %               If the last input argument is a parameter list (plist) it is used.
0016 %               The following parameters are recognised.
0017 %
0018 % PARAMETERS:   'Win'   - a specwin window object [default: Kaiser -200dB psll]
0019 %               'Olap' - segment percent overlap [default: taken from window function]
0020 %               'Nfft' - number of samples in each fft [default: length of input data]
0021 %                   A string value containing the variable 'fs' can
0022 %                   also be used, e.g., plist('Nfft', '2*fs')
0023 %               'Scale' - one of
0024 %                           'ASD' - amplitude spectral density
0025 %                           'PSD' - power spectral density [default]
0026 %                           'AS'  - amplitude spectrum
0027 %                           'PS'  - power spectrum
0028 %               'Order' - order of segment detrending
0029 %                   -1 - no detrending
0030 %                   0 - subtract mean [default]
0031 %                   1 - subtract linear fit
0032 %                   N - subtract fit of polynomial, order N
0033 %
0034 % M-FILE INFO:  Get information about this methods by calling
0035 %               >> ao.getInfo('pwelch')
0036 %
0037 %               Get information about a specified set-plist by calling:
0038 %               >> ao.getInfo('pwelch', 'None')
0039 %
0040 % VERSION:      $Id: pwelch.m,v 1.14 2008/08/07 13:00:39 ingo Exp $
0041 %
0042 % HISTORY:      07-02-2007 M Hewitson
0043 %               Creation
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 function varargout = pwelch(varargin)
0047     warning('! This method is deprecated and will be removed from future releases of the
0048 toolbox. Use ao/psd instead.')
0049     if nargin == 0
0050         psd(varargin{:})
0051     else
0052         varargout{:} = psd(varargin{:});
0053     end
0054 end
0055 % END
0056

```

rms

PURPOSE ^

RMS Calculate RMS deviation from spectrum

SYNOPSIS ^

function varargout = rms(varargin)

DESCRIPTION ^

```
RMS Calculate RMS deviation from spectrum
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RMS Calculate RMS deviation from spectrum

CALL:      b=rms(a)

INPUTS:    a - input analysis object containing spectrum

OUTPUTS:    b - analysis object containing RMS deviation

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('rms')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('rms', 'None')

VERSION:    $Id: rms.m,v 1.9 2008/09/05 11:05:29 ingo Exp $

NOTE:       Taken from code by: 1998.05.25      Masaki Ando

HISTORY:    12-02-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [sqrt](#) SQRT computes the square root of the data in the AO.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % RMS Calculate RMS deviation from spectrum
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RMS Calculate RMS deviation from spectrum
0005 %
0006 % CALL:      b=rms(a)
0007 %
0008 % INPUTS:    a - input analysis object containing spectrum
0009 %
```



```

0010 % OUTPUTS:      b - analysis object containing RMS deviation
0011 %
0012 % M-FILE INFO:  Get information about this methods by calling
0013 %               >> ao.getInfo('rms')
0014 %
0015 %               Get information about a specified set-plist by calling:
0016 %               >> ao.getInfo('rms', 'None')
0017 %
0018 % VERSION:      $Id: rms.m,v 1.9 2008/09/05 11:05:29 ingo Exp $
0019 %
0020 % NOTE:          Taken from code by: 1998.05.25      Masaki Ando
0021 %
0022 % HISTORY:       12-02-07 M Hewitson
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 function varargout = rms(varargin)
0028
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 import utils.const.*
0036 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0037
0038 % Collect input variable names
0039 in_names = cell(size(varargin));
0040 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0041
0042 % Collect all AOs and plists
0043 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0044
0045 % Decide on a deep copy or a modify
0046 bs = copy(as, nargout);
0047
0048 % Loop over input AOs
0049 for j=1:numel(bs)
0050     % check input data
0051     if isa(bs(j).data, 'fsdata')
0052         % get data
0053         f = bs(j).data.getX;
0054         spe = [f.' bs(j).data.getY.'];
0055         % start and end frequencies
0056         s = f(1);
0057         e = f(end);
0058         % compute integrated rms
0059         l1=spe(:,1)>=s;
0060         sp=spe(l1,:);
0061         l2=sp(:,1)<=e;
0062         sp=sp(l2,:);
0063         si=size(sp);
0064         li=si(1,1);
0065         freq=sp(:,1);
0066         sp2=sp(:,2).^2;
0067         ms=sp2;
0068         for i= li-1 :-1: 1
0069             ms(i)=ms(i+1)+(sp2(i+1)+sp2(i))*(freq(i+1)-freq(i))/2;
0070         end
0071         % set data
0072         bs(j).setXY(freq, sqrt(ms), 'internal');
0073         % set name
0074         bs(j).setName(sprintf('RMS(%s)', ao_invars{j}), 'internal');
0075         % Add history
0076         bs(j).addHistory(getInfo, plist, ao_invars(j), bs(j).hist);
0077     else
0078         warning('!!! Skipping AO %s - it's not an frequency series.', ao_invars{j});
0079     end
0080 end
0081
0082 % Set output
0083 if nargout > 0
0084     varargout{1} = bs;
0085 end
0086 end
0087
0088 %-----
0089 % Get Info Object
0090 %-----
0091 function ii = getInfo(varargin)
0092     if nargin == 1 && strcmpi(varargin{1}, 'None')
0093         sets = {};
0094         pl = [];
0095     else

```

```
0096     sets = {'Default'};
0097     pl     = getDefaultPlist;
0098 end
0099 % Build info object
0100 ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: rms.m,v 1.9
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0101 end
0102
0103 %-----
0104 % Get Default Plist
0105 %-----
0106 function pl_default = getDefaultPlist()
0107     pl_default = plist();
0108 end
0109 % END
0110
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

select

PURPOSE ^

SELECT select particular samples from the input AOs and return new AOs with only those samples.

SYNOPSIS ^

function varargout = select(varargin)

DESCRIPTION ^

```
SELECT select particular samples from the input AOs and return new AOs with only those
samples.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SELECT select particular samples from the input AOs and return
new AOs with only those samples.

CALL:      b = select(a, [1 2 3 4]);
           b = select(a, pl)

PARAMETERS: 'samples' - a list of samples to select

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('select')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('select', 'None')

VERSION:    $Id: select.m,v 1.26 2008/09/05 11:05:29 ingo Exp $

HISTORY:    14-05-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [t0](#) T0 Get the data property 't0'.
-  [x](#) X Get the data property 'x'.

This function is called by:

-  [lxspecc](#) LXSPEC performs log-scale cross-spectral analysis of various forms.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % SELECT select particular samples from the input AOs and return new AOs with only
those samples.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SELECT select particular samples from the input AOs and return
0005 %               new AOs with only those samples.
0006 %
0007 % CALL:         b = select(a, [1 2 3 4]);
0008 %               b = select(a, pl)
0009 %
0010 % PARAMETERS: 'samples' - a list of samples to select
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %               >> ao.getInfo('select')
0014 %
0015 %               Get information about a specified set-plist by calling:
0016 %               >> ao.getInfo('select', 'None')
0017 %
0018 % VERSION:      $Id: select.m,v 1.26 2008/09/05 11:05:29 ingo Exp $
0019 %
0020 % HISTORY:      14-05-07 M Hewitson
0021 %               Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = select(varargin)
0026
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 import utils.const.*
0034 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0035
0036 % Collect input variable names
0037 in_names = cell(size(varargin));
0038 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0039
0040 % Collect all AOs and plists
0041 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042 [pl, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0043
0044 % Decide on a deep copy or a modify
0045 bs = copy(as, nargout);
0046
0047 % Combine plists
0048 pl = combine(pl, getDefaultPlist);
0049
0050 % Check for samples in arguments
0051 samples = [];
0052 for j=1:numel(rest)
0053     if isnumeric(rest{j})
0054         samples = [samples rest{j}];
0055     end
0056 end
0057
0058 % Get sample selection from plist
0059 samples = [samples find(pl, 'samples')];
0060
0061 % Loop over input AOs
0062 for j=1:numel(bs)
0063     if isa(bs(j).data, 'cdata')
0064         bs(j).setY(bs(j).data.y(samples), 'internal');
0065     else
0066         % Get x
0067         x = bs(j).data.getX;
0068         % set new samples
0069         bs(j).setXY(x(samples), bs(j).data.y(samples), 'internal');
0070         % if this is tsdata, we can collapse it again, maybe
0071         if isa(bs(j).data, 'tsdata')
0072             [fs,t0,fitted] = tsdata.fitfs(bs(j).data.x);
0073             if ~fitted
0074                 bs(j).data.collapseX();
0075             end
0076         end
0077     end
0078     % Set AO name
0079     bs(j).setName(sprintf('select(%s)', ao_invars{j}), 'internal');
0080     % Add history
0081     bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0082 end
0083
0084 % Set outputs
0085 if nargout > 0

```

```
0086     varargout{1} = bs;
0087     end
0088
0089 end
0090
0091 %-----
0092 % Get Info Object
0093 %-----
0094 function ii = getInfo(varargin)
0095     if nargin == 1 && strcmpi(varargin{1}, 'None')
0096         sets = {};
0097         pl = [];
0098     else
0099         sets = {'Default'};
0100         pl = getDefaultPlist;
0101     end
0102     % Build info object
0103     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: select.m,v 1.26
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0104 end
0105
0106 %-----
0107 % Get Default Plist
0108 %-----
0109 function pl_default = getDefaultPlist()
0110     pl_default = plist('samples', []);
0111 end
0112
0113 % END
```

smoother

PURPOSE ^

SMOOTHER smooths a given series of data points using the specified method.

SYNOPSIS ^

function varargout = smoother(varargin)

DESCRIPTION ^

```
SMOOTHER smooths a given series of data points using the specified method.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SMOOTHER smooths a given series of data points using
             the specified method.

CALL:       b = smoother(a, pl)

PARAMETERS: width  - the width of the smoothing filter [default: 20 samples]
             hc     - a cutoff to throw away outliers (0-1) [default: 0.8]
             method - the smoothing method:
                       'median' [default]
                       'mean', 'min', 'max', 'mode'

M-FILE INFO: Get information about this methods by calling
             >> ao.getInfo('smoother')

             Get information about a specified set-plist by calling:
             >> ao.getInfo('smoother', 'None')

VERSION:     $Id: smoother.m,v 1.12 2008/09/05 11:05:29 ingo Exp $

HISTORY:     02-03-08 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [sort](#) SORT the values in the AO.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [firwhiten](#) FIRWHITEN whitens the input time-series by building an FIR whitening filter.
-  [linedetect](#) LINEDETECT find spectral lines in the ao/fsdata objects.

SUBFUNCTIONS ^

-  [function](#) ii = getInfo(varargin)
-  [function](#) pl = getDefaultPlist()
-  [function](#) ys = smooth(y, bw, hc, method)

SOURCE CODE ^

```

0001 % SMOOTHER smooths a given series of data points using the specified method.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SMOOTHER smooths a given series of data points using
0005 %               the specified method.
0006 %
0007 % CALL:         b = smoother(a, pl)
0008 %
0009 % PARAMETERS:   width  - the width of the smoothing filter [default: 20 samples]
0010 %               hc     - a cutoff to throw away outliers (0-1) [default: 0.8]
0011 %               method - the smoothing method:
0012 %                       'median' [default]
0013 %                       'mean', 'min', 'max', 'mode'
0014 %
0015 % M-FILE INFO:  Get information about this methods by calling
0016 %               >> ao.getInfo('smoother')
0017 %
0018 %               Get information about a specified set-plist by calling:
0019 %               >> ao.getInfo('smoother', 'None')
0020 %
0021 % VERSION:      $Id: smoother.m,v 1.12 2008/09/05 11:05:29 ingo Exp $
0022 %
0023 % HISTORY:      02-03-08 M Hewitson
0024 %               Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0027
0028 function varargout = smoother(varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 import utils.const.*
0037 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0038
0039 % Collect input variable names
0040 in_names = cell(size(varargin));
0041 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0042
0043 % Collect all AOs and plists
0044 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0045 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0046
0047 % Decide on a deep copy or a modify
0048 bs = copy(as, nargout);
0049
0050 % combine plists
0051 pl = combine(pl, getDefaultPlist());
0052
0053 % Get parameters from plist
0054 bw = find(pl, 'width');
0055 hc = find(pl, 'hc');
0056 method = find(pl, 'method');
0057
0058 % check the method
0059 if ~strcmp(method, 'median') && ...
0060     ~strcmp(method, 'mean') && ...
0061     ~strcmp(method, 'min') && ...
0062     ~strcmp(method, 'max') && ...
0063     ~strcmp(method, 'mode')
0064     help(mfilename)
0065     error('### Unknown smoothing method');
0066 end
0067
0068 % Loop over input AOs
0069 for j=1:length(bs)
0070     utils.helper.msg(msg.PROC1, 'smoothing %s', bs(j).name);
0071     if strcmp(method, 'median')
0072         bs(j).data.setY(ltpda_smoother(bs(j).data.getY, bw, hc, method), 'internal');
0073     else
0074         bs(j).data.setY(smooth(bs(j).data.getY, bw, hc, method), 'internal');
0075     end
0076     % Add history
0077     bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
0078     % set name
0079     bs(j).setName(sprintf('smoother(%s)', ao_invars{j}), 'internal');
0080 end
0081
0082 if nargout > 0
0083     varargout{1} = bs;
0084 end
0085 end

```

```

0086
0087 %-----
0088 % Get Info Object
0089 %-----
0090 function ii = getInfo(varargin)
0091     if nargin == 1 && strcmpi(varargin{1}, 'None')
0092         sets = {};
0093         pl = [];
0094     else
0095         sets = {'Default'};
0096         pl = getDefaultPlist;
0097     end
0098     % Build info object
0099     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: smoother.m,v
1.12 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0100 end
0101
0102 %-----
0103 % Get Default Plist
0104 %-----
0105 function pl = getDefaultPlist()
0106     pl = plist('width', 20, 'hc', 0.8, 'method', 'median');
0107 end
0108
0109 %-----
0110 % smooth data
0111 function ys = smooth(y, bw, hc, method)
0112     N = length(y);
0113     ys = zeros(size(y));
0114     for kk=1:N
0115         if mod(kk, 1000)==0
0116             disp(sprintf(' -- smoothed %06d samples', kk));
0117         end
0118         % Determine the interval we are looking in
0119         interval = kk-bw/2:kk+bw/2;
0120         interval(interval<=0)=1;
0121         interval(interval>N)=N;
0122         % calculate median value of interval
0123         % after throwing away outliers
0124         trial = sort(y(interval));
0125         b = round(hc*length(trial));
0126         eval(sprintf('ys(kk) = %s(trial(1:b));', method));
0127     end
0128 end
0129 % END
0130

```


spectrogram

PURPOSE ^

SPECTROGRAM computes a spectrogram of the given ao/tsdata.

SYNOPSIS ^

function varargout = spectrogram(varargin)

DESCRIPTION ^

```
SPECTROGRAM computes a spectrogram of the given ao/tsdata.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SPECTROGRAM computes a spectrogram of the given ao/tsdata
              using MATLAB's spectrogram function.

CALL:        b = spectrogram(a, pl)

PARAMETERS:

    'win'     - a specwin object [default: Kaiser -200dB psll]
    'Nolap'   - segment overlap [default: taken from window function]
    'Nfft'    - number of samples in each short fourier transform
                  [default: sample rate of data]

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('spectrogram')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('spectrogram', 'set')

VERSION:      $Id: spectrogram.m,v 1.10 2008/09/05 14:15:40 hewitson Exp $

HISTORY: 12-03-07 M Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [spectrogram](#) SPECTROGRAM computes a spectrogram of the given ao/tsdata.

This function is called by:

-  [spectrogram](#) SPECTROGRAM computes a spectrogram of the given ao/tsdata.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SPECTROGRAM computes a spectrogram of the given ao/tsdata.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0003 %
0004 % DESCRIPTION: SPECTROGRAM computes a spectrogram of the given ao/tsdata
0005 %                using MATLAB's spectrogram function.
0006 %
0007 % CALL:         b = spectrogram(a, pl)
0008 %
0009 % PARAMETERS:
0010 %
0011 %             'Win'   - a specwin object [default: Kaiser -200dB psll]
0012 %             'Nolap' - segment overlap [default: taken from window function]
0013 %             'Nfft'  - number of samples in each short fourier transform
0014 %                      [default: sample rate of data]
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 %               >> ao.getInfo('spectrogram')
0018 %
0019 %               Get information about a specified set-plist by calling:
0020 %               >> ao.getInfo('spectrogram', 'set')
0021 %
0022 % VERSION:      $Id: spectrogram.m,v 1.10 2008/09/05 14:15:40 hewitson Exp $
0023 %
0024 % HISTORY: 12-03-07 M Hewitson
0025 %           Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = spectrogram(varargin)
0030
0031 bs = [];
0032
0033 %% Check if this is a call for parameters
0034 if utils.helper.isinfocall(varargin{:})
0035     varargout{1} = getInfo(varargin{3});
0036     return
0037 end
0038
0039 import utils.const.*
0040 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0041
0042 if nargin == 0
0043     error('### spectrogram cannot be used as a modifier. Please give an output
variable.');
```

```

0044 end
0045
0046 % Collect input variable names
0047 in_names = cell(size(varargin));
0048 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0049
0050 % Collect all AOs
0051 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0052 [ps, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0053
0054 % Process parameters
0055 pl = combine(ps, getDefaultPlist);
0056
0057 % Check input analysis object
0058 for j=1:numel(as)
0059     a = as(j);
0060
0061     % Get settings for this AO
0062     nfft = find(pl, 'Nfft');
0063     if isempty(nfft) || nfft < 0
0064         nfft = a.data.fs;
0065     end
0066     win = find(pl, 'Win');
0067     if length(win.win) < nfft
0068         switch win.type
0069             case {'Kaiser', 'Flattop'}
0070                 win = specwin(win.type, nfft, win.psll);
0071             otherwise
0072                 win = specwin(win.type, nfft);
0073         end
0074         utils.helper.msg(msg.PROC1, 'reset window to %s(%d)', strrep(win.type, '_', '\_'),
length(win.win));
0075     end
0076     nolap = find(pl, 'Nolap');
0077     if isempty(nolap) || nolap < 0
0078         nolap = floor(win.rov*nfft/100);
0079     end
0080
0081 % Process data
0082 [S, F, T, P] = spectrogram(a.data.y, win.win, nolap, nfft, a.data.fs);
0083
0084 % Make output AO
0085 do = xyzdata(T, F, P);
0086 do.setXunits('s');
```

```

0087     do.setYunits('Hz');
0088     do.setZunits(a.data.yunits^2 / unit('Hz'));
0089
0090     b = ao(do);
0091     b.setName(sprintf('spectgram(%s)', ao_invars{j}), 'internal');
0092     b.addHistory(getInfo(), pl, cellstr(ao_invars{j}), a.hist);
0093
0094     % add to output
0095     bs = [bs b];
0096 end
0097
0098 % Reshape the ouput to the same size of the input
0099 bs = reshape(bs, size(as));
0100
0101 varargout{1} = bs;
0102 end
0103
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %                                     Local Functions
0106 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0107
0108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109 %
0110 % FUNCTION:      getInfo
0111 %
0112 % DESCRIPTION:  Get Info Object
0113 %
0114 % HISTORY:      11-07-07 M Hewitson
0115 %               Creation.
0116 %
0117 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0118
0119 function ii = getInfo(varargin)
0120     if nargin == 1 && strcmpi(varargin{1}, 'None')
0121         sets = {};
0122         pl = [];
0123     else
0124         sets = {'Default'};
0125         pl = getDefaultPlist;
0126     end
0127     % Build info object
0128     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: spectrogram.m,v
1.10 2008/09/05 14:15:40 hewitson Exp $', sets, pl);
0129 end
0130
0131 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0132 %
0133 % FUNCTION:      getDefaultPlist
0134 %
0135 % DESCRIPTION:  Get Default Plist
0136 %
0137 % HISTORY:      11-07-07 M Hewitson
0138 %               Creation.
0139 %
0140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141
0142 function plo = getDefaultPlist()
0143     plo = plist('Win', getappdata(0, 'ltpda_default_spectral_window'), 'Nolap', -1,
'Nfft', -1);
0144 end
0145

```

spikecleaning

PURPOSE ^

spikecleaning detects and corrects possible spikes in analysis objects

SYNOPSIS ^

function varargout=spikecleaning(varargin)

DESCRIPTION ^

```
spikecleaning detects and corrects possible spikes in analysis objects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SPIKECLEANING detects spikes in the temperature data and
             replaces them by artificial values depending on the method chosen ('random',
             'mean', 'previous').
             Spikes are defined as singular samples with an (absolute) value
             higher than kspike times the standard deviation of the high-pass
             filtered (IIR filter) input AO.

CALL:        b = spikecleaning(a1, a2, ..., an, pl)

INPUTS:      aN - a list of analysis objects
              pl - parameter list

OUTPUTS:      b - a list of analysis objects with "spike values" removed
              and corrected

PARAMETERES: 'kspike' - set the kspike value. High values imply
                  not correction of relative low amplitude spike
                  [default: 3.3]
              'method' - method used to replace the spike value: 'random',
                  'mean', 'previous' [default:random]
              'fc' - frequency cut-off of the IIR filter [default: 0.025]
              'order' - order of the IIR filter [default: 2]
              'ripple' - specify pass/stop-band ripple for bandpass
                  and bandreject filters
                  <<default: 0.5>>

VERSION:      $Id: spikecleaning.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:      24-12-2007 J Sanjuan
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [ao](#) AO analysis object class constructor.
-  [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [mean](#) MEAN computes the mean value of the data in the AO.
-  [std](#) STD computes the standard deviation of the data in the AO.

This function is called by:

SUBFUNCTIONS ^

 `function ii = getInfo(varargin)`
 `function plo = getDefaultPlist()`

SOURCE CODE

```

0001 % spikecleaning detects and corrects possible spikes in analysis objects
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SPIKECLEANING detects spikes in the temperature data and
0005 %             replaces them by artificial values depending on the method chosen
0006 %             ('random',
0007 %             'mean', 'previous').
0008 %             Spikes are defined as singular samples with an (absolute) value
0009 %             higher than kspike times the standard deviation of the high-pass
0010 %             filtered (IIR filter) input A0.
0011 % CALL:      b = spikecleaning(a1, a2, ..., an, pl)
0012 %
0013 % INPUTS:    aN - a list of analysis objects
0014 %            pl - parameter list
0015 %
0016 % OUTPUTS:   b - a list of analysis objects with "spike values" removed
0017 %             and corrected
0018 %
0019 % PARAMETERES: 'kspike' - set the kspike value. High values imply
0020 %               not correction of relative low amplitude spike
0021 %               [default: 3.3]
0022 %               'method' - method used to replace the spike value: 'random',
0023 %               'mean', 'previous' [default: random]
0024 %               'fc' - frequency cut-off of the IIR filter [default: 0.025]
0025 %               'order' - order of the IIR filter [default: 2]
0026 %               'ripple' - specify pass/stop-band ripple for bandpass
0027 %               and bandreject filters
0028 %               <<default: 0.5>>
0029 %
0030 % VERSION:   $Id: spikecleaning.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0031 %
0032 % HISTORY:   24-12-2007 J Sanjuan
0033 %             Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 function varargout=spikecleaning(varargin)
0038
0039 %%% Check if this is a call for parameters
0040 if utils.helper.isinfocall(varargin{:})
0041     varargout{1} = getInfo(varargin{3});
0042     return
0043 end
0044
0045 if nargin == 0
0046     error('### cat cannot be used as a modifier. Please give an output variable.');
```

```

0077 fs = plist();
0078 fs.append('fs', d.fs);
0079 pls.combine(fs); % determination of the sampling frequency of the
input AO
0080
0081 % high-pass filtering data
0082 xfiltered = filtfilt(a, miir(pls));
0083
0084 % standard deviation of the filtered data is calculated
0085 nxfiltered = find(abs(xfiltered) < kspike*std(xfiltered));
0086
0087 xfiltered_2 = xfiltered.data.y(nxfiltered);
0088
0089 std_xfiltered_2 = std(xfiltered_2);
0090
0091 % spikes vector position is determined
0092 nspike = find(abs(xfiltered) > kspike*std_xfiltered_2);
0093
0094 % substitution of spike values starts here
0095 xcleaned = a.data.y;
0096 for j=1:length(nspike)
0097     if nspike(j) <=2 % just in case a spike is detected in the 1st or 2nd sample
0098         xcleaned(nspike(j)) = mean(xcleaned(1:50));
0099     else
0100         if strcmp(method, 'random') % spike is substituted by a random value:
N(0,std_xfiltered)
0101         xcleaned(nspike(j)) = xcleaned(nspike(j)-1) + randn(1)*std_xfiltered_2;
0102         elseif strcmp(method, 'mean') % spike is substituted by the mean if the two
previous values
0103         xcleaned(nspike(j)) = (xcleaned(nspike(j)-1) + xcleaned(nspike(j)-2))/2;
0104         elseif strcmp(method, 'previous') % spike is substituted by the pervious value
0105         xcleaned(nspike(j)) = xcleaned(nspike(j)-1);
0106     end
0107 end
0108 end
0109
0110 % create new output tsdata
0111 ts = tsdata(xcleaned, d.fs);
0112 ts.setYunits(d.yunits);
0113 ts.setXunits(d.xunits);
0114
0115 % % create new output history
0116 % h = history(ALGONAME, VERSION, pls, a.hist);
0117 % h = set(h, 'invars', invars);
0118
0119 % make output analysis object
0120 b = ao(ts);
0121 b.setName(sprintf('spikecleaning(%s)', ao_invars{i}), 'internal');
0122 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%');
0123 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%');
0124 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%');
0125 %warning('%%%%%%%%%% Uncomment the following line (add Hsitory) %%%%%%%%%');
0126 b.addHistory(getInfo(), pls, ao_invars(i), as(i).hist);
0127
0128 % add to output array
0129 bo = [bo b];
0130
0131 varargout{1} = bo;
0132 end
0133 end
0134
0135 %%%%%%%%%%%
0136 % Local Functions
0137 %%%%%%%%%%%
0138 %%%%%%%%%%%
0139 %%%%%%%%%%%
0140 %
0141 % FUNCTION: getInfo
0142 %
0143 % DESCRIPTION: Get Info Object
0144 %
0145 % HISTORY: 11-07-07 M Hewitson
0146 % Creation.
0147 %
0148 %%%%%%%%%%%
0149
0150 function ii = getInfo(varargin)
0151 if nargin == 1 && strcmpi(varargin{1}, 'None')
0152     sets = {};
0153     pl = [];
0154 else
0155     sets = {'Default'};
0156     pl = getDefaultPlist();
0157 end
0158 % Build info object
0159 ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id:

```

```
spikecleaning.m,v 1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0160 end
0161
0162 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0163 %
0164 % FUNCTION:      getDefaultPlist
0165 %
0166 % DESCRIPTION:  Get Default Plist
0167 %
0168 % HISTORY:      11-07-07 M Hewitson
0169 %                Creation.
0170 %
0171 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0172
0173 function plo = getDefaultPlist()
0174
0175     plo = plist(
0176         'kspike', 3.3, ...
0177         'fc', 0.025, ...
0178         'order', 2, ...
0179         'ripple', 0.5, ...
0180         'method', 'random');
0181 end
0182
```


split

PURPOSE ^

SPLIT split an analysis object into the specified segments.

SYNOPSIS ^

function varargout = **split**(varargin)

DESCRIPTION ^

```
SPLIT split an analysis object into the specified segments.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SPLIT split an analysis object into the specified segments.

CALL:      b = split(a, pl)

INPUTS:    a - input analysis object
            pl - input parameter list (see below for parameters)

OUTPUTS:    b - array of analysis objects

PARAMETERS: <key>      <value>      <description>
            'split_type' 'times'      split the ao into time segments
            'frequencies' 'frequencies' split the ao into frequencies segments
            'samples'      'samples'      split the ao into sample segments
            'chunks'       'chunks'       split the ao into chunks
            'duration'     'duration'     select a duration of a tsdata

Necessary for the individual split types:
            <split type> <key>      <description>
            'times'      'times'      an array of start/stop times to
end                                                split by. A negative stop time is taken from the
from                                                of the vector, e.g., [10 -10] removes 10 seconds
                                                the beginning and end of the
                                                vector. An end time of 0
                                                indicates the end of the vector.
            'frequencies' 'frequencies' an array of start/stop frequencies to split by
            'samples'     'samples'     an array of start/stop samples to split by
            'chunks'      'N'           split into N contiguous pieces
            'interval'    'start_time', 'stop_time'
                        - start/stop time can be either a string or a time
                        object (if a string, 'format_str' must be
                        specified or specify the strings in UTC time format)
            'interval'    'start_time', 'duration'
                        - start time and the duration can be either a string
                        or a time object (if a string, 'format_str' must be
                        specified or specify the strings in UTC time format)
            'interval'    'timespan'
                        - the start/end time are specified in the time span
                        object.

The UTC time format is: 'yyyy-mm-dd HH:MM:SS'

If more than one splitting method is specified, the priority
goes like the list above.

The time vector in the output A0 retains the original
time values (i.e. it doesn't start from zero).

The splitting is done as  s<=t<e.

Arrays of start/stop values should be like: [s1 e1 s2 e2 ....]

EXAMPLES:  1.) Split method by frequency. Get the values from 10-100 Hz
            pl = plist('split_type', 'frequencies', ...
                        'frequencies', [10 100]);
            ao_new = split(a1, pl);

            2.) Split method by time.
```



```

Get the values from 0.0 to 1.0 Seconds AND from 1.0 to 2.5 seconds
pl = plist('split_type', 'times', ...
           'times',      [0.0 1.0 1.0 2.5]);
ao_new = split(a1, pl);

3.) Split method by samples.
Get the samples from 0 to 50 AND from 150 to 200.
pl = plist('split_type', 'samples', ...
           'samples',    [0 50 150 200]);
ao_new = split(a1, pl);

4.1) Select an interval with strings
pl = plist('split_type', 'interval', ...
           'start_time', '14:00:01', ...
           'end_time',   '14:00:02');
ao_new = split(a1, pl);

pl = plist('split_type', 'interval', ...
           'start_time', '14:00:01', ...
           'duration',   '00:00:02');
ao_new = split(a1, pl);

Select an interval with milliseconds
pl = plist('split_type', 'interval', ...
           'start_time', 5000, ...
           'end_time',   7000);
ao_new = split(a1, pl);

4.2) Select an interval with time objects
pl = plist('split_type', 'interval', ...
           'start_time', time('14:00:01'), ...
           'end_time',   time('14:00:03'));
ao_new = split(a1, pl);

pl = plist('split_type', 'interval', ...
           'start_time', time(5000), ...
           'duration',   time(2000));
ao_new = split(a1, pl);

4.3) Select an interval with a time span object
pl = plist('split_type', 'interval', ...
           'timespan',   timespan('14:00:00', '14:00:05'));
ao_new = split(a1, pl);

```

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('split')

Get information about a specified set-plist by calling:
 >> ao.getInfo('split', 'Default')

VERSION: \$Id: split.m,v 1.56 2008/09/05 11:05:29 ingo Exp \$

HISTORY: 02-03-07 M Hewitson
 Creation.

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **ao** AO analysis object class constructor.
-  **copy** COPY makes a (deep) copy of the input AOs.
-  **find** FIND particular samples that satisfy the input query and return a new AO.
-  **split** SPLIT split an analysis object into the specified segments.
-  **t0** T0 Get the data property 't0'.
-  **times** TIMES implements times operator for analysis objects.
-  **x** X Get the data property 'x'.
-  **y** Y Get the data property 'y'.

This function is called by:

-  **consolidate** CONSOLIDATE resamples all input AOs onto the same time grid.
-  **split** SPLIT split an analysis object into the specified segments.

SUBFUNCTIONS [^]

 function ii = getInfo(varargin)
 function plo = getDefaultPlist(set)

SOURCE CODE [^]

```

0001 % SPLIT split an analysis object into the specified segments.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SPLIT split an analysis object into the specified segments.
0005 %
0006 % CALL:          b = split(a, pl)
0007 %
0008 % INPUTS:        a - input analysis object
0009 %                pl - input parameter list (see below for parameters)
0010 %
0011 % OUTPUTS:       b - array of analysis objects
0012 %
0013 % PARAMETERS: <key>      <value>      <description>
0014 %              'split_type'  'times'        split the ao into time segments
0015 %                      'frequencies' split the ao into frequencies segments
0016 %                      'samples'    split the ao into sample segments
0017 %                      'chunks'     split the ao into chunks
0018 %                      'duration'   select a duration of a tsdata
0019 %
0020 % Necessary for the individual split types:
0021 %              <split_type> <key>      <description>
0022 %              'times'      'times'      an array of start/stop times to
0023 %              the end      split by. A negative stop time is taken from
0024 %              seconds from of the vector, e.g., [10 -10] removes 10
0025 %                          the beginning and end of the
0026 %                          vector. An end time of 0
0027 %                          indicates the end of the vector.
0028 %              'frequencies' 'frequencies' an array of start/stop frequencies to split
0029 %              by          'samples'      an array of start/stop samples to split by
0030 %              'chunks'     'N'          split into N contiguous pieces
0031 %              'interval'   'start_time', 'stop_time'
0032 %                          - start/stop time can be either a string or a time
0033 %                          object (if a string, 'format_str' must be
0034 %                          specified or specify the strings in UTC time format)
0035 %              'interval'   'start_time', 'duration'
0036 %                          - start time and the duration can be either a string
0037 %                          or a time object (if a string, 'format_str' must be
0038 %                          specified or specify the strings in UTC time format)
0039 %              'interval'   'timespan'
0040 %                          - the start/end time are specified in the time span
0041 %                          object.
0042 %
0043 %              The UTC time format is: 'yyyy-mm-dd HH:MM:SS'
0044 %
0045 %              If more than onesplitting method is specified, the priority
0046 %              goes like the list above.
0047 %
0048 %              The time vector in the output AO retains the original
0049 %              time values (i.e. it doesn't start from zero).
0050 %
0051 %              The splitting is done as  s<=t<e.
0052 %
0053 %              Arrays of start/stop values should be like: [s1 e1 s2 e2 ....]
0054 %
0055 % EXAMPLES:  1.) Split method by frequency. Get the values from 10-100 Hz
0056 %              pl = plist('split_type', 'frequencies', ...
0057 %                      'frequencies', [10 100]);
0058 %              ao_new = split(a1, pl);
0059 %
0060 %              2.) Split method by time.
0061 %              Get the values from 0.0 to 1.0 Seconds AND from 1.0 to 2.5 seconds
0062 %              pl = plist('split_type', 'times', ...
0063 %                      'times', [0.0 1.0 1.0 2.5]);
0064 %              ao_new = split(a1, pl);
0065 %
0066 %              3.) Split method by samples.
0067 %              Get the samples from 0 to 50 AND from 150 to 200.
0068 %              pl = plist('split_type', 'samples', ...
0069 %                      'samples', [0 50 150 200]);
0070 %              ao_new = split(a1, pl);
0071 %

```

```

0072 %      4.1) Select an interval with strings
0073 %          pl = plist('split_type', 'interval', ...
0074 %                  'start_time', '14:00:01', ...
0075 %                  'end_time', '14:00:02');
0076 %          ao_new = split(a1, pl);
0077 %
0078 %          pl = plist('split_type', 'interval', ...
0079 %                  'start_time', '14:00:01', ...
0080 %                  'duration', '00:00:02');
0081 %          ao_new = split(a1, pl);
0082 %
0083 %          Select an interval with milliseconds
0084 %          pl = plist('split_type', 'interval', ...
0085 %                  'start_time', 5000, ...
0086 %                  'end_time', 7000);
0087 %          ao_new = split(a1, pl);
0088 %
0089 %      4.2) Select an interval with time objects
0090 %          pl = plist('split_type', 'interval', ...
0091 %                  'start_time', time('14:00:01'), ...
0092 %                  'end_time', time('14:00:03'));
0093 %          ao_new = split(a1, pl);
0094 %
0095 %          pl = plist('split_type', 'interval', ...
0096 %                  'start_time', time(5000), ...
0097 %                  'duration', time(2000));
0098 %          ao_new = split(a1, pl);
0099 %
0100 %      4.3) Select an interval with a time span object
0101 %          pl = plist('split_type', 'interval', ...
0102 %                  'timespan', timespan('14:00:00', '14:00:05'));
0103 %          ao_new = split(a1, pl);
0104 %
0105 % M-FILE INFO: Get information about this methods by calling
0106 %             >> ao.getInfo('split')
0107 %
0108 %             Get information about a specified set-plist by calling:
0109 %             >> ao.getInfo('split', 'Default')
0110 %
0111 % VERSION:      $Id: split.m,v 1.56 2008/09/05 11:05:29 ingo Exp $
0112 %
0113 % HISTORY:      02-03-07 M Hewitson
0114 %              Creation.
0115 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116 %
0117 function varargout = split(varargin)
0118 %
0119 %%% Check if this is a call for parameters
0120 if utils.helper.isinfocall(varargin{:})
0121     varargout{1} = getInfo(varargin{3});
0122     return
0123 end
0124 %
0125 import utils.const.*
0126 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0127 %
0128 if nargin == 0
0129     error('### split cannot be used as a modifier. Please give an output variable.');
```

```

0158     if isempty(split_type)
0159         error('### please specify the key ''split_type'' in the parameter list');
0160     end
0161
0162     % look at input data
0163     x = as(j).data.getX;
0164     y = as(j).data.getY;
0165
0166     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0167     %                               splitting by time or frequency                               %
0168     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0169     if strcmpi(split_type, 'times') || strcmpi(split_type, 'frequencies')
0170
0171         times = find(pl, 'times');
0172         frequencies = find(pl, 'frequencies');
0173
0174         if ~isempty(times)
0175             utils.helper.msg(msg.PROC1, 'splitting by time');
0176             split_x_axis.type = 'times';
0177             split_x_axis.value = times;
0178             if ~isa(as(j).data, 'tsdata')
0179                 error('### I can only split time-series by times');
0180             end
0181         else
0182             utils.helper.msg(msg.PROC1, 'splitting by frequency');
0183             split_x_axis.type = 'frequencies';
0184             split_x_axis.value = frequencies;
0185             if ~isa(as(j).data, 'fsdata')
0186                 error('### I can only split frequency-series by frequencies');
0187             end
0188         end
0189
0190         % examine time list
0191         ntimes = length(split_x_axis.value);
0192         if mod(ntimes, 2) ~= 0
0193             error('### please specify a start and stop for each interval.')
0194         end
0195         % go over each interval now
0196         for i=1:2:ntimes
0197             is = split_x_axis.value(i);
0198             ie = split_x_axis.value(i+1);
0199             if ie < 0 % indicates count from end
0200                 ie = as(j).data.nsecs + ie;
0201                 if ie < is
0202                     error('### End time is before the start time.');

```

```

0243 npairs = length(samples);
0244 if mod(npairs, 2) ~= 0
0245     error('### please specify a start and stop for each interval.')
0246 end
0247 % check data
0248 if length(x) ~= length(y) && ~isempty(x)
0249     error('### Something is wrong with the x/y vectors. I can't split this
data. ');
0250 end
0251 % go over each interval now
0252 utils.helper.msg(msg.PROC1, sprintf('Split: \f'));
0253 c = 1;
0254 dtmp = copy(as(j).data,1);
0255 dtmp.setXY([],[]);
0256 for i=1:2:npairs
0257     utils.helper.msg(msg.PROC1, sprintf('\b%03d ', round(c)));
0258     if mod(c,11) == 0 && c < npairs
0259         utils.helper.msg(msg.PROC1, sprintf('Chunk: \f'));
0260     end
0261     is = samples(i);
0262     ie = samples(i+1);
0263     % create new output data
0264     if ~isempty(x)
0265         x1 = x(is:ie);
0266     else
0267         x1 = [];
0268     end
0269     y1 = y(is:ie);
0270     % subtract offset of x for tsdata
0271     % - we record this info by chaging to
0272     d = copy(as(j).data, 1);
0273     if isa(d, 'tsdata')
0274         d.setX(x1-x1(1));
0275     else
0276         d.setX(x1);
0277     end
0278     d.setY(y1);
0279     % set t0,nsecs if this is tsdata
0280     if isa(d, 'tsdata')
0281         t0 = dtmp.t0 + x1(1);
0282         d.setT0(t0);
0283         d.setNsecs(length(y1) ./ dtmp.fs);
0284     end
0285     % Make AO
0286     b = ao(d);
0287     % create new output history
0288     b.addHistory(getInfo, plist('samples', [is ie]), ao_invars(j), as(j).hist);
0289     % set name
0290     b.setName(sprintf('split(%s)[%d]', ao_invars{j},c), 'internal');
0291     % Add to output array
0292     bo = [bo b];
0293     c = c + 1;
0294 end
0295
0296 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0297 %                                splitting into chunks                                %
0298 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0299 elseif strcmpi(split_type, 'chunks')
0300     N = find(pl, 'N');
0301     utils.helper.msg(msg.PROC1, 'splitting into %d chunks', N);
0302     % chunk size
0303     csize = floor(length(y)/N);
0304     % generate list of indices
0305     idx = [];
0306     for kk=1:N
0307         is = 1+(kk-1)*csize;
0308         ie = csize*kk;
0309         idx = [idx is ie];
0310     end
0311     % one call to split with these samples
0312     b = split(as, plist('split_type', 'samples', 'samples', idx));
0313     bo = [bo b];
0314
0315 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0316 %                                splitting into interval                                %
0317 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0318 elseif strcmpi(split_type, 'interval')
0319
0320     % txt = '#####';
0321     % error('%s\n### Please, check this part before using it due to the fact
that the objects are handles.\n%s', txt, txt)
0322
0323     %% get values from the parameter list
0324     duration = find(pl, 'duration');
0325     start_time = find(pl, 'start_time');
0326     end_time = find(pl, 'end_time');

```



```

0327     format_str = find(pl, 'format_str');
0328     timespan    = find(pl, 'timespan');
0329
0330     %%% Some checks
0331     if (~isempty(start_time) || ~isempty(end_time)) && ~isempty(timespan)
0332         error('### Please specify only a timespan and not additionally the start/end
time');
0333     end
0334
0335     %%% Convert the start_time into a time object
0336     if ~isempty(start_time) && ~isa(start_time, 'time')
0337         start_time = time(start_time);
0338     end
0339
0340     %%% Convert the end_time into a time object
0341     if ~isempty(end_time) && ~isa(end_time, 'time')
0342         end_time = time(end_time);
0343     end
0344
0345     %%% Convert the duration
0346     if ~isempty(end_time) && ~isempty(duration)
0347         error('### Please specify only a duration or an end time');
0348     end
0349     if ~isempty(duration) && ~isa(duration, 'time')
0350         duration = time(duration);
0351         end_time = start_time + duration;
0352         end_time = time(end_time);
0353     end
0354
0355     %%% Set start/end time with a timespan object
0356     if ~isempty(timespan)
0357         if ~isa(timespan, 'timespan')
0358             error('### The timespan must be a timespan object')
0359         end
0360         if ~isempty(start_time) || ~isempty(end_time)
0361             error('### Please specify only a timespan OR a start/end time');
0362         end
0363
0364         start_time = timespan.startT;
0365         end_time    = timespan.endT;
0366     end
0367
0368     %%% Set the end_time to the same timezone as the start_time
0369     end_time.setTimezone(start_time.timezone);
0370
0371     t0_time = as(j).data.t0;
0372     t1      = as(j).data.getX(1);
0373
0374     %%% Compute the start/end time
0375     ts = (start_time.utc_epoch_milli-t0_time.utc_epoch_milli+t1)/1000;
0376     te = (end_time.utc_epoch_milli-t0_time.utc_epoch_milli+t1)/1000;
0377
0378     idx = find(x >= ts & x < te);
0379     %%% create new output data
0380     d = copy(as(j).data,1);
0381     d.setXY(x(idx), y(idx));
0382     % Set nsecs for tsdata
0383     if isa(d, 'tsdata')
0384         if ~isempty(d.x)
0385             d.setNsecs(d.x(end) - d.x(1) + 1/d.fs);
0386         else
0387             d.setNsecs(0);
0388         end
0389     end
0390     % Make AO
0391     b = ao(d);
0392     % create new output history
0393     b.addHistory(getInfo, pl, ao_invars(j), as(j).hist);
0394     % set name
0395     b.name = sprintf('split(%s)', ao_invars{j});
0396     % Add to output array
0397     bo = [bo b];
0398
0399     end
0400 end % for j = 1:numel(as)
0401
0402 % Set output
0403 varargout{1} = bo;
0404 end
0405
0406 %-----
0407 % Get Info Object
0408 %-----
0409
0410 function ii = getInfo(varargin)
0411     if nargin == 1 && strcmpi(varargin{1}, 'None')

```

```

0412     sets = {};
0413     pls = [];
0414     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0415         sets{1} = varargin{1};
0416         pls = getDefaultPlist(sets{1});
0417     else
0418         sets = {...
0419             'Default', ...
0420             'By times', ...
0421             'By frequencies', ...
0422             'By samples', ...
0423             'By chunks', ...
0424             'By interval start/end', ...
0425             'By interval start/duration', ...
0426             'By interval timespan'};
0427         pls = [];
0428         for kk=1:numel(sets)
0429             pls = [pls getDefaultPlist(sets{kk})];
0430         end
0431     end
0432     % Build info object
0433     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: split.m,v 1.56
2008/09/05 11:05:29 ingo Exp $', sets, pls);
0434 end
0435
0436 %-----
0437 % Get Default Plist
0438 %-----
0439 function plo = getDefaultPlist(set)
0440     switch set
0441     case 'Default'
0442         plo = plist('split_type', 'times', 'times', []);
0443     case 'By times'
0444         plo = plist('split_type', 'times', 'times', []);
0445     case 'By frequencies'
0446         plo = plist('split_type', 'frequencies', 'frequencies', []);
0447     case 'By samples'
0448         plo = plist('split_type', 'samples', 'samples', []);
0449     case 'By chunks'
0450         plo = plist('split_type', 'chunks', 'N', []);
0451     case 'By interval start/end'
0452         plo = plist('split_type', 'interval', 'start_time', time(0), 'end_time', time(0));
0453     case 'By interval start/duration'
0454         plo = plist('split_type', 'interval', 'start_time', time(0), 'duration', time(0));
0455     case 'By interval timespan'
0456         plo = plist('split_type', 'interval', 'timespan', timespan(0,0));
0457     otherwise
0458         error('### Unknown parameter set [%s].', set);
0459     end
0460 end
0461
0462

```

tfe

PURPOSE ^

TFE makes transfer function estimates of the time-series objects.

SYNOPSIS ^

function varargout = tfe(varargin)

DESCRIPTION ^

```
TFE makes transfer function estimates of the time-series objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION:  TFE makes transfer function estimates of the time-series
               objects in the input analysis objects. Transfer functions are
               computed using MATLAB's tfestimate (>> help tfestimate).

CALL:        b = tfe(a1,a2,a3,...,pl)

INPUTS:      b      - output analysis objects
               aN     - input analysis objects (at least two)
               pl     - input parameter list

OUTPUTS:     The output matrix b contains NxN transfer functions estimates (all
               possible pairs of input AOs).

               If the last input argument is a parameter list (plist) it is used.
               The following parameters are recognised.

PARAMETERS:  'Win'    - a specwin window object [default: Kaiser -200dB psll]
               'Olap'  - segment percent overlap [default: taken from window function]
               'Nfft'  - number of samples in each fft [default: length of input data]
               Order   - order of detrending:
                       -1 - no detrending
                       0 - subtract mean [default]
                       1 - subtract linear fit
                       N - subtract fit of polynomial, order N

M-FILE INFO: Get information about this methods by calling
               >> ao.getInfo('tfe')

               Get information about a specified set-plist by calling:
               >> ao.getInfo('tfe', 'None')

VERSION:     $Id: tfe.m,v 1.6 2008/09/05 11:05:29 ingo Exp $

HISTORY:     07-02-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^


```

0001 % TFE makes transfer function estimates of the time-series objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TFE makes transfer function estimates of the time-series
0005 %               objects in the input analysis objects. Transfer functions are
0006 %               computed using MATLAB's tfestimate (>> help tfestimate).
0007 %
0008 % CALL:         b = tfe(a1,a2,a3,...,pl)
0009 %
0010 % INPUTS:       b      - output analysis objects
0011 %               aN     - input analysis objects (at least two)
0012 %               pl     - input parameter list
0013 %
0014 % OUTPUTS:      The output matrix b contains NxN transfer functions estimates (all
0015 %               possible pairs of input AOs).
0016 %
0017 %               If the last input argument is a parameter list (plist) it is used.
0018 %               The following parameters are recognised.
0019 %
0020 % PARAMETERS:   'Win'   - a specwin window object [default: Kaiser -200dB psll]
0021 %               'Olap'  - segment percent overlap [default: taken from window function]
0022 %               'Nfft'  - number of samples in each fft [default: length of input data]
0023 %               Order  - order of detrending:
0024 %                   -1 - no detrending
0025 %                   0 - subtract mean [default]
0026 %                   1 - subtract linear fit
0027 %                   N - subtract fit of polynomial, order N
0028 %
0029 % M-FILE INFO:  Get information about this methods by calling
0030 %               >> ao.getInfo('tfe')
0031 %
0032 %               Get information about a specified set-plist by calling:
0033 %               >> ao.getInfo('tfe', 'None')
0034 %
0035 % VERSION:      $Id: tfe.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0036 %
0037 % HISTORY:      07-02-2007 M Hewitson
0038 %               Creation
0039 %
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041
0042 function varargout = tfe(varargin)
0043
0044 % Check if this is a call for parameters
0045 if utils.helper.isinfocall(varargin{:})
0046     varargout{1} = getInfo(varargin{3});
0047     return
0048 end
0049
0050 import utils.const.*
0051 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0052
0053 % Collect input variable names
0054 in_names = cell(size(varargin));
0055 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0056
0057 % Collect all AOs and plists
0058 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0059 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0060
0061 % combine plists
0062 pl = combine(pl, getDefaultPlist());
0063
0064 varargout{1} = ao.xspec(as, pl, 'tfe', getInfo, ao_invars);
0065 end
0066
0067 %-----
0068 % Get Info Object
0069 %-----
0070 function ii = getInfo(varargin)
0071     if nargin == 1 && strcmpi(varargin{1}, 'None')
0072         sets = {};
0073         pl = [];
0074     else
0075         sets = {'Default'};
0076         pl = getDefaultPlist;
0077     end
0078 % Build info object
0079 ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: tfe.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0080 end
0081
0082 %-----
0083 % Get Default Plist
0084 %-----

```

```
0085 function pl = getDefaultPlist()
0086     pl = plist('Nfft', -1, ...
0087         'win', getappdata(0, 'ltpda_default_spectral_window'), ...
0088         'Olap', -1, ...
0089         'Order', 0);
0090 end
0091
0092 % END
0093
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

timedomainfit

PURPOSE ^

TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.

SYNOPSIS ^

function varargout = timedomainfit(varargin)

DESCRIPTION ^

TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
%%%

DESCRIPTION: TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.

CALL: [x stdx mse S] = timedomainfit(a1,a2,a3,..., pl)

INPUTS: asN - a list of analysis objects containing the data to be fitted.
The first input analysis object that contains time-series data is the target object. A linear combination of all other time-series AOs is fit to this target.
pl - a parameter list

OUTPUTS: x - cdata type analysis object containing the fitting coefficients
stdx - estimated standard errors of x
mse - mean squared error
S - estimated covariance matrix of X

M-FILE INFO: Get information about this methods by calling
>> class.getInfo('timedomainfit')

Get information about a specified set-plist by calling:
>> class.getInfo('timedomainfit', 'set')

VERSION: \$Id: timedomainfit.m,v 1.6 2008/09/05 11:05:29 ingo Exp \$

HISTORY: 01-03-2007 M Hewitson
Creation
%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [x](#) X Get the data property 'x'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
```

```

0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to
0005 % a target time-series AO.
0006 %
0007 % CALL: [x stdx mse S] = timedomainfit(a1,a2,a3,..., pl)
0008 %
0009 % INPUTS: asN - a list of analysis objects containing the data to be fitted.
0010 % The first input analysis object that contains time-series
0011 % data is the target object. A linear combination of all other
0012 % time-series AOs is fit to this target.
0013 % pl - a parameter list
0014 %
0015 % OUTPUTS: x - cdata type analysis object containing the fitting
0016 % coefficients
0017 % stdx - estimated standard errors of x
0018 % mse - mean squared error
0019 % S - estimated covariance matrix
0020 % of X
0021 % M-FILE INFO: Get information about this methods by calling
0022 % >> class.getInfo('timedomainfit')
0023 %
0024 % Get information about a specified set-plist by calling:
0025 % >> class.getInfo('timedomainfit', 'set')
0026 %
0027 % VERSION: $Id: timedomainfit.m,v 1.6 2008/09/05 11:05:29 ingo Exp $
0028 %
0029 % HISTORY: 01-03-2007 M Hewitson
0030 % Creation
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032
0033 function varargout = timedomainfit(varargin)
0034
0035 %% Check if this is a call for parameters
0036 if utils.helper.isinfocall(varargin{:})
0037     varargout{1} = getInfo(varargin{3});
0038     return
0039 end
0040
0041 import utils.const.*
0042 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0043
0044 %% Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 %% Collect all AOs
0049 [as, ao_invars, rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0050 [pli, pl_invars, rest] = utils.helper.collect_objects(rest, 'plist', in_names);
0051
0052 %% Decide on a deep copy or a modify
0053 %% REMARK: If you create a new AO (call the constructor) then
0054 %% it is not necessary to copy the input-AOs !!!!!!!!!!!!!!!!!!!!!!!
0055 bs = copy(as, nargin);
0056
0057 %% Combine plists
0058 pl = combine(pli, getDefaultPlist);
0059 % k = find(pl, 'bin');
0060 %% go through analysis objects
0061 x = [];
0062 for kk = 1:numel(bs)
0063     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064     if ~isreal(bs(kk).data.y)
0065         error('### I only work with real time-series at the moment.');

```

```
0088 %      bs(kk).data.setY(some_new_data);
0089 %      bs(kk).data.setFs(new_fs);
0090 %      bs(kk).data.setXunits(new_xunits);
0091 %      bd(kk).setName('my name');
0092
0093 %%% Set Name
0094 %      bs(kk).setName('new name', 'internal');
0095
0096 %%% Add History
0097 %      bs(kk).addHistory(getInfo('None'), pl, ao_invars(kk), bs(kk).hist);
0098
0099
0100
0101 %%% Reshape the output to the same size of the input
0102 varargout{1} = coef;
0103 varargout{2} = stdx;
0104 varargout{3} = mse;
0105 varargout{4} = S;
0106 end
0107
0108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109 %                                Local Functions                                %
0110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111
0112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113 %
0114 % FUNCTION:      getInfo
0115 %
0116 % DESCRIPTION:  Get Info Object
0117 %
0118 % HISTORY:      11-07-07 M Hewitson
0119 %                Creation.
0120 %
0121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122
0123 function ii = getInfo(varargin)
0124     if nargin == 1 && strcmpi(varargin{1}, 'None')
0125         sets = {};
0126         pl   = [];
0127     else
0128         sets = {'Default'};
0129         pl   = getDefaultPlist;
0130     end
0131     % Build info object
0132     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id:
timedomainfit.m,v 1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0133 end
0134
0135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0136 %
0137 % FUNCTION:      getDefaultPlist
0138 %
0139 % DESCRIPTION:  Get Default Plist
0140 %
0141 % HISTORY:      11-07-07 M Hewitson
0142 %                Creation.
0143 %
0144 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0145
0146 function plo = getDefaultPlist()
0147     plo = plist();
0148 end
0149
0150
```

xcorr

PURPOSE ^

XCORR makes cross-correlation estimates of the time-series

SYNOPSIS ^

function varargout = xcorr(varargin)

DESCRIPTION ^

```
XCORR makes cross-correlation estimates of the time-series
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: XCORR makes cross-correlation estimates of the time-series
              objects in the input analysis objects. The cross-correlation is
              computed using MATLAB's xcorr (>> help xcorr).

CALL:        b = xcorr(a1,a2,a3,...,pl)

INPUTS:      b      - output analysis objects
              aN     - input analysis objects (at least two)
              pl     - input parameter list

              The function makes correlation estimates between a1 and all other
              input AOs. Therefore, if the input argument list contains N analysis
              objects, the output a will contain N-1 CPSD estimates.

              If only on AO is input, the auto-correlation is computed.

              If the last input argument is a parameter list (plist) it is used.
              The following parameters are recognised.

PARAMETERS:  'MaxLag'   - compute over range of lags -MaxLag to MaxLag [default: M-1]
              'Scale'   - normalisation of the correlation. Choose from:
                  'biased' - scales the raw cross-correlation by 1/M.
                  'unbiased' - scales the raw correlation by 1/(M-abs(lags)).
                  'coeff'  - normalizes the sequence so that the auto-correlations
                           at zero lag are identically 1.0.
                  'none'   - no scaling (this is the default).

              M is the length of longest input vector. If one vector is shorted,
              it is zero padded.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('xcorr')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('xcorr', 'None')

VERSION:     $Id: xcorr.m,v 1.5 2008/09/05 14:15:40 hewitson Exp $

HISTORY:     07-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [findFsMax](#) findFsMax Returns the max Fs of a set of AOs
-  [findShortestVector](#) findShortestVector Returns the length of the shortest vector in samples

-  **fs** FS Get the data property 'fs'.
-  **len** LEN overloads the length operator for Analysis objects. Length of the data samples.
-  **xcorr** XCORR makes cross-correlation estimates of the time-series

This function is called by:

-  **xcorr** XCORR makes cross-correlation estimates of the time-series

SUBFUNCTIONS [^]

-  **function** lmin = findShortestVector(as)
-  **function** fs = findFsMax(as)
-  **function** ii = getInfo(varargin)
-  **function** plo = getDefaultPlist()

SOURCE CODE [^]

```

0001 % XCORR makes cross-correlation estimates of the time-series
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: XCORR makes cross-correlation estimates of the time-series
0005 %                objects in the input analysis objects. The cross-correlation is
0006 %                computed using MATLAB's xcorr (>> help xcorr).
0007 %
0008 % CALL:         b = xcorr(a1,a2,a3,...,pl)
0009 %
0010 % INPUTS:      b      - output analysis objects
0011 %              aN     - input analysis objects (at least two)
0012 %              pl     - input parameter list
0013 %
0014 %              The function makes correlation estimates between a1 and all other
0015 %              input AOs. Therefore, if the input argument list contains N analysis
0016 %              objects, the output a will contain N-1 CPSD estimates.
0017 %
0018 %              If only on AO is input, the auto-correlation is computed.
0019 %
0020 %              If the last input argument is a parameter list (plist) it is used.
0021 %              The following parameters are recognised.
0022 %
0023 % PARAMETERS:  'MaxLag'   - compute over range of lags -MaxLag to MaxLag [default: M-
0024 %                  1]
0025 %              'Scale'    - normalisation of the correlation. Choose from:
0026 %                  'biased' - scales the raw cross-correlation by 1/M.
0027 %                  'unbiased' - scales the raw correlation by 1/(M-abs(lags)).
0028 %                  'coeff'   - normalizes the sequence so that the auto-correlations
0029 %                  at zero lag are identically 1.0.
0030 %                  'none'    - no scaling (this is the default).
0031 %
0032 %              M is the length of longest input vector. If one vector is shorted,
0033 %              it is zero padded.
0034 % M-FILE INFO:  Get information about this methods by calling
0035 %              >> ao.getInfo('xcorr')
0036 %
0037 %              Get information about a specified set-plist by calling:
0038 %              >> ao.getInfo('xcorr', 'None')
0039 %
0040 % VERSION:     $Id: xcorr.m,v 1.5 2008/09/05 14:15:40 hewitson Exp $
0041 %
0042 % HISTORY:     07-02-2007 M Hewitson
0043 %              Creation
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 function varargout = xcorr(varargin)
0048 %
0049 % Check if this is a call for parameters
0050 if utils.helper.isinfocall(varargin{:})
0051     varargout{1} = getInfo(varargin{3});
0052     return
0053 end
0054 %
0055 import utils.const.*
0056 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0057

```



```

0058 % Collect input variable names
0059 in_names = cell(size(varargin));
0060 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0061
0062 % Collect all AOs and plists
0063 [as, invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0064 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0065
0066 % combine plists
0067 pl = combine(pl, getDefaultPlist());
0068
0069 na = numel(as);
0070 if na < 2
0071     error('### XCORR needs at least two AOs to cross-correlate.');
```

0072 end

0073

0074 %----- Resample all AOs

0075 copies = zeros(size(as));

0076

0077 fsmax = findFsMax(as);

0078 fspl = plist(param('fsout', fsmax));

0079 for i=1:na

0080 % check this is a time-series object

0081 if ~isa(as(i).data, 'tsdata')

0082 error('### ltpda_xspec requires tsdata (time-series) inputs.');

0083 end

0084 % Check Fs

0085 if as(i).data.fs ~= fsmax

0086 utils.helper.msg(msg.PROC1, 'resampling AO %s to %f Hz', as(i).name, fsmax);

0087 % Make a deep copy so we don't

0088 % affect the original input data

0089 as(i) = copy(as(i), 1);

0090 copies(i) = 1;

0091 as(i).resample(fspl);

0092 end

0093 end

0094

0095 %----- Truncate all vectors

0096

0097 % Get shortest vector

0098 disp('*** Truncating all vectors...');

0099 lmin = findShortestVector(as);

0100 nsecs = lmin / fsmax;

0101 for i=1:na

0102 if len(as(i)) ~= lmin

0103 utils.helper.msg(msg.PROC2, 'truncating AO %s to %d secs', as(i).name, nsecs);

0104 % do we already have a copy?

0105 if ~copies(i)

0106 % Make a deep copy so we don't

0107 % affect the original input data

0108 as(i) = copy(as(i), 1);

0109 copies(i) = 1;

0110 end

0111 as(i).select(1:lmin);

0112 end

0113 end

0114

0115 %----- check input parameters

0116

0117 % Maximum lag for Xcorr

0118 MaxLag = find(pl, 'MaxLag');

0119

0120 % Scale for Xcorr

0121 scale = find(pl, 'Scale');

0122

0123 % Loop over input AOs

0124 bs(na,na) = ao;

0125 for j=1:na

0126 for k=1:na

0127 % ----- Make Xspec estimate

0128

0129 % Compute cross-correlation estimates using XCORR

0130 if MaxLag == -1

0131 MaxLag = len(as(j));

0132 end

0133 [c, lags] = xcorr(as(j).data.getY, as(k).data.getY, MaxLag, scale);

0134

0135 % Keep the data shape of the first input AO

0136 if size(as(1).data.y,1) == 1

0137 c = c.';

0138 end

0139

0140 % create new output fsdata

0141 xy = xydata(lags./fsmax, c);

0142 xy.setXunits('s');

0143 xy.setYunits('');


```

0144
0145 %----- create new output history
0146
0147 % we need to get the input histories in the same order as the inputs
0148 % to this function call, not in the order of the input to tfestimate;
0149 % otherwise the resulting matrix on a 'create from history' will be
0150 % mirrored.
0151 if j<k
0152     bs(j,k).addHistory(getInfo('None'), pl, [invars(j) invars(k)], [as(j).hist,
as(k).hist]);
0153 else
0154     bs(j,k).addHistory(getInfo('None'), pl, [invars(j) invars(k)], [as(k).hist,
as(j).hist]);
0155 end
0156
0157 % make output analysis object
0158 bs(j,k).data = xy;
0159 % set name
0160 bs(j,k).setName(sprintf('xcorr(%s->%s)', invars{j}, invars{k}), 'internal');
0161 end % End second loop over AOs
0162 end % End first loop over AOs
0163
0164 % Set outputs
0165 varargout{1} = bs;
0166 end
0167
0168 %-----
0169 % Returns the length of the shortest vector in samples
0170 function lmin = findShortestVector(as)
0171     lmin = 1e20;
0172     for j=1:length(as)
0173         if len(as(j)) < lmin
0174             lmin = len(as(j));
0175         end
0176     end
0177 end
0178 %-----
0179 % Returns the max Fs of a set of AOs
0180 function fs = findFsMax(as)
0181     fs = 0;
0182     for j=1:length(as)
0183         a = as(j);
0184         if a.data.fs > fs
0185             fs = a.data.fs;
0186         end
0187     end
0188 end
0189
0190 %-----
0191 % Get Info Object
0192 %-----
0193 function ii = getInfo(varargin)
0194     if nargin == 1 && strcmpi(varargin{1}, 'None')
0195         sets = {};
0196         pl = [];
0197     else
0198         sets = {'Default'};
0199         pl = getDefaultPlist;
0200     end
0201     % Build info object
0202     ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: xcorr.m,v 1.5
2008/09/05 14:15:40 hewitson Exp $', sets, pl);
0203 end
0204
0205 %-----
0206 % Get Default Plist
0207 %-----
0208 function plo = getDefaultPlist()
0209
0210     plo = plist('MaxLag', -1, 'Scale', 'none');
0211
0212 end

```

acos

PURPOSE ^

ACOS overloads the acos method for Analysis objects.

SYNOPSIS ^

function varargout = acos (varargin)

DESCRIPTION ^

ACOS overloads the acos method for Analysis objects.
%%%

DESCRIPTION: ACOS overloads the acos operator for Analysis objects.
Inverse cosine result in radians.
ACOS(ao) is the cosine of the elements of ao.data.

CALL: ao_out = acos(ao_in);
 ao_out = acos(ao_in, pl);
 ao_out = acos(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('acos')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('acos', 'None')

VERSION: \$Id: acos.m,v 1.19 2008/09/05 11:05:28 ingo Exp \$

HISTORY: 07-05-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ACOS overloads the acos method for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ACOS overloads the acos operator for Analysis objects.
0005 %               Inverse cosine result in radians.
0006 %               ACOS(ao) is the cosine of the elements of ao.data.
0007 %
0008 % CALL:        ao_out = acos(ao_in);
0009 %               ao_out = acos(ao_in, pl);
```

```

0010 %          ao_out = acos(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % PARAMETERS:  see help for data2D/applymethod for additional parameters
0013 %
0014 % M-FILE INFO:  Get information about this methods by calling
0015 %               >> ao.getInfo('acos')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('acos', 'None')
0019 %
0020 % VERSION:      $Id: acos.m,v 1.19 2008/09/05 11:05:28 ingo Exp $
0021 %
0022 % HISTORY:      07-05-2007 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%
0026
0027 function varargout = acos (varargin)
0028
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs
0040 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0041 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargout);
0045
0046 % Apply method to all AOs
0047 applymethod(bs, ao_invars, 'acos', pl, getDefaultPlist, getInfo);
0048
0049 % Set output
0050 if nargout == 1
0051     varargout{1} = bs;
0052 end
0053 end
0054
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl   = [];
0062     else
0063         sets = {'Default'};
0064         pl   = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.trig, '$Id: acos.m,v 1.19
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0068 end
0069
0070 %-----
0071 % Get Default Plist
0072 %-----
0073 function pl_default = getDefaultPlist()
0074     pl_default = plist('axis', 'y');
0075 end
0076
0077

```

asin

PURPOSE ^

ASIN overloads the asin method for Analysis objects.

SYNOPSIS ^

function varargout = asin (varargin)

DESCRIPTION ^

ASIN overloads the asin method for Analysis objects.
%%%

DESCRIPTION: ASIN overloads the asin operator for Analysis objects.
Inverse sine result in radians.
ASIN(ao) is the sine of the elements of ao.data.

CALL: ao_out = asin(ao_in);
 ao_out = asin(ao_in, pl);
 ao_out = asin(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('asin')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('asin', 'None')

VERSION: \$Id: asin.m,v 1.16 2008/09/05 11:05:28 ingo Exp \$

HISTORY: 07-05-2007 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ASIN overloads the asin method for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ASIN overloads the asin operator for Analysis objects.
0005 %              Inverse sine result in radians.
0006 %              ASIN(ao) is the sine of the elements of ao.data.
0007 %
0008 % CALL:        ao_out = asin(ao_in);
0009 %              ao_out = asin(ao_in, pl);
```

```

0010 %          ao_out = asin(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % PARAMETERS: see help for data2D/applymethod for additional parameters
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> ao.getInfo('asin')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('asin', 'None')
0019 %
0020 % VERSION:      $Id: asin.m,v 1.16 2008/09/05 11:05:28 ingo Exp $
0021 %
0022 % HISTORY:      07-05-2007 Diepholz
0023 %                Creation
0024 %
0025 %%%%%%%%%%%
0026
0027 function varargout = asin (varargin)
0028
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs
0040 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0041 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargout);
0045
0046 % Apply method to all AOs
0047 applymethod(bs, ao_invars, 'asin', pl, getDefaultPlist, getInfo);
0048
0049 % Set output
0050 if nargout == 1
0051     varargout{1} = bs;
0052 end
0053 end
0054
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl   = [];
0062     else
0063         sets = {'Default'};
0064         pl   = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.trig, '$Id: asin.m,v 1.16
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0068 end
0069
0070 %-----
0071 % Get Default Plist
0072 %-----
0073 function pl_default = getDefaultPlist()
0074     pl_default = plist('axis', 'y');
0075 end
0076

```

atan

PURPOSE ^

ATAN overloads the atan method for Analysis objects.

SYNOPSIS ^

function varargout = atan (varargin)

DESCRIPTION ^

ATAN overloads the atan method for Analysis objects.
%%%

DESCRIPTION: ATAN overloads the atan operator for Analysis objects.
Inverse sine result in radians.
ATAN(ao) is the sine of the elements of ao.data.

CALL: ao_out = atan(ao_in);
 ao_out = atan(ao_in, pl);
 ao_out = atan(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
 >> ao.getInfo('atan')

 Get information about a specified set-plist by calling:
 >> ao.getInfo('atan', 'None')

VERSION: \$Id: atan.m,v 1.16 2008/09/05 11:05:28 ingo Exp \$

HISTORY: 07-05-2007 Diepholz
 Creation
%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ATAN overloads the atan method for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ATAN overloads the atan operator for Analysis objects.
0005 %              Inverse sine result in radians.
0006 %              ATAN(ao) is the sine of the elements of ao.data.
0007 %
0008 % CALL:      ao_out = atan(ao_in);
0009 %           ao_out = atan(ao_in, pl);
```

```

0010 %          ao_out = atan(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % PARAMETERS: see help for data2D/applymethod for additional parameters
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> ao.getInfo('atan')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> ao.getInfo('atan', 'None')
0019 %
0020 % VERSION:      $Id: atan.m,v 1.16 2008/09/05 11:05:28 ingo Exp $
0021 %
0022 % HISTORY:      07-05-2007 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%
0026
0027 function varargout = atan (varargin)
0028
0029 % Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs
0040 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0041 pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargout);
0045
0046 % Apply method to all AOs
0047 applymethod(bs, ao_invars, 'atan', pl, getDefaultPlist, getInfo);
0048
0049 % Set output
0050 if nargout == 1
0051     varargout{1} = bs;
0052 end
0053 end
0054
0055 %-----
0056 % Get Info Object
0057 %-----
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl   = [];
0062     else
0063         sets = {'Default'};
0064         pl   = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'ao', '', utils.const.categories.trig, '$Id: atan.m,v 1.16
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0068 end
0069
0070 %-----
0071 % Get Default Plist
0072 %-----
0073 function pl_default = getDefaultPlist()
0074     pl_default = plist('axis', 'y');
0075 end
0076
0077

```

atan2

PURPOSE ^

ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse tangent.

SYNOPSIS ^

function varargout = atan2 (varargin)

DESCRIPTION ^

```
ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse tangent.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: ATAN2 overloads the atan2 operator for Analysis objects.
             Four quadrant inverse tangent.

REMARK:      The data-object of the output A0 is the same as the data-object
             of the first A0 input. The result of the atan2 will be copied to
             the y values of this data-object.

CALL:        ao_out = atan2(ao1, ao2);
             ao_out = atan2(ao1, ao2);
             ao_out = atan2(ao_vec1, ao_vec2);

M-FILE INFO: Get information about this methods by calling
             >> ao.getInfo('atan2')

             Get information about a specified set-plist by calling:
             >> ao.getInfo('atan2', 'None')

VERSION:     $Id: atan2.m,v 1.8 2008/09/05 11:05:28 ingo Exp $

HISTORY:     16-05-2008 Diepholz
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse
0002 % tangent.
0003 %
0004 % DESCRIPTION: ATAN2 overloads the atan2 operator for Analysis objects.
0005 %             Four quadrant inverse tangent.
0006 %
0007 % REMARK:      The data-object of the output A0 is the same as the data-object
```



```

0008 %           of the first A0 input. The result of the atan2 will be copied to
0009 %           the y values of this data-object.
0010 %
0011 % CALL:      ao_out = atan2(ao1, ao2);
0012 %           ao_out = atan2(ao1, ao2);
0013 %           ao_out = atan2(ao_vec1, ao_vec2);
0014 %
0015 % M-FILE INFO: Get information about this methods by calling
0016 %           >> ao.getInfo('atan2')
0017 %
0018 %           Get information about a specified set-plist by calling:
0019 %           >> ao.getInfo('atan2', 'None')
0020 %
0021 % VERSION:    $Id: atan2.m,v 1.8 2008/09/05 11:05:28 ingo Exp $
0022 %
0023 % HISTORY:    16-05-2008 Diepholz
0024 %           Creation
0025 %
0026 %%%%%%%%%%%
0027 %
0028 function varargout = atan2 (varargin)
0029
0030 % Check if this is a call for parameters
0031 if utils.helper.isinfocall(varargin{:})
0032     varargout{1} = getInfo(varargin{3});
0033     return
0034 end
0035
0036 % Collect input variable names
0037 in_names = cell(size(varargin));
0038 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0039
0040 % Collect all AOs
0041 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0042
0043 % Decide on a deep copy or a modify
0044 bs = copy(as, nargout);
0045
0046 % Check input arguments number
0047 if length(bs) ~= 2
0048     error ('### Incorrect inputs. Please enter 2 AOs');
0049 end
0050
0051 % Apply method to all AOs
0052 bs = applyoperator(bs, ao_invars, 'atan2', 'atan2', getDefaultPlist, getInfo);
0053
0054 % Set output
0055 if nargout == 1
0056     varargout{1} = bs;
0057 end
0058 end
0059
0060 %-----
0061 % Get Info Object
0062 %-----
0063 function ii = getInfo(varargin)
0064     if nargin == 1 && strcmpi(varargin{1}, 'None')
0065         sets = {};
0066         pl = [];
0067     else
0068         sets = {'Default'};
0069         pl = getDefaultPlist;
0070     end
0071     % Build info object
0072     ii = minfo(mfilename, 'ao', '', utils.const.categories.trig, '$Id: atan2.m,v 1.8
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0073 end
0074
0075 %-----
0076 % Get Default Plist
0077 %-----
0078 function pl_default = getDefaultPlist()
0079     pl_default = plist();
0080 end
0081

```

COS

PURPOSE ^

COS overloads the cos operator for Analysis objects. Cosine of argument in radians.

SYNOPSIS ^

function varargout = cos (varargin)

DESCRIPTION ^

COS overloads the cos operator for Analysis objects. Cosine of argument in radians.
%%%

DESCRIPTION: COS overloads the cos operator for Analysis objects.
Cosine of argument in radians.
COS(ao) is the cosine of the elements of ao.data.

CALL: ao_out = cos(ao_in);
ao_out = cos(ao_in, pl);
ao_out = cos(ao1, pl1, ao_vector, ao_matrix, pl2);

POSSIBLE VALUES: ao_in = [ao2 ao3]
ao_in = ao_vector
ao_in = ao_matrix

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
>> ao.getInfo('cos')

Get information about a specified set-plist by calling:
>> ao.getInfo('cos', 'None')

HISTORY: 07-05-2007 Diepholz
Creation
%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % COS overloads the cos operator for Analysis objects. Cosine of argument in radians.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COS overloads the cos operator for Analysis objects.
0005 %               Cosine of argument in radians.
0006 %               COS(ao) is the cosine of the elements of ao.data.
0007 %
```

```

0008 % CALL: ao_out = cos(ao_in);
0009 %         ao_out = cos(ao_in, pl);
0010 %         ao_out = cos(ao1, pl1, ao_vector, ao_matrix, pl2);
0011 %
0012 % POSSIBLE VALUES: ao_in = [ao2 ao3]
0013 %                     ao_in = ao_vector
0014 %                     ao_in = ao_matrix
0015 %
0016 % PARAMETERS: see help for data2D/applymethod for additional parameters
0017 %
0018 % M-FILE INFO: Get information about this methods by calling
0019 %               >> ao.getInfo('cos')
0020 %
0021 %               Get information about a specified set-plist by calling:
0022 %               >> ao.getInfo('cos', 'None')
0023 %
0024 % HISTORY: 07-05-2007 Diepholz
0025 %           Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = cos (varargin)
0030
0031 % Check if this is a call for parameters
0032 if utils.helper.isinfocall(varargin{:})
0033     varargout{1} = getInfo(varargin{3});
0034     return
0035 end
0036
0037 % Collect input variable names
0038 in_names = cell(size(varargin));
0039 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0040
0041 % Collect all AOs
0042 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0043 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0044
0045 % Decide on a deep copy or a modify
0046 bs = copy(as, nargout);
0047
0048 % Combine plists
0049 pl = combine(pl, getDefaultPlist);
0050
0051 % Apply method to all AOs
0052 applymethod(bs, ao_invars, 'cos', pl, getDefaultPlist, getInfo);
0053
0054 % Reshape the ouput to the same size of the input
0055 if nargout > 0
0056     varargout{1} = bs;
0057 end
0058 end
0059
0060 %-----
0061 % Get Info Object
0062 %-----
0063 function ii = getInfo(varargin)
0064
0065 if nargin == 1 && strcmpi(varargin{1}, 'None')
0066     sets = {};
0067     pl = [];
0068 else
0069     sets = {'Default'};
0070     pl = getDefaultPlist;
0071 end
0072 % Build info object
0073 ii = minfo(mfilename, 'ao', '', utils.const.categories.trig, '$Id: cos.m,v 1.15
2008/09/05 11:05:28 ingo Exp $', sets, pl);
0074 end
0075
0076 %-----
0077 % Get Default Plist
0078 %-----
0079 function pl_default = getDefaultPlist()
0080 pl_default = plist('axis', 'y');
0081 end
0082
0083

```

sin

PURPOSE ^

SIN overloads the sin method for Analysis objects.

SYNOPSIS ^

function varargout = sin (varargin)

DESCRIPTION ^

```
SIN overloads the sin method for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SIN overloads the sin operator for Analysis objects.

CALL:
    ao_out = sin(ao_in);
    ao_out = sin(ao_in, pl);
    ao_out = sin(ao1, pl1, ao_vector, ao_matrix, pl2);

PARAMETERS: see help for data2D/applymethod for additional parameters

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('sin')

    Get information about a specified set-plist by calling:
    >> ao.getInfo('sin', 'set')

VERSION:    $Id: sin.m,v 1.23 2008/09/05 11:05:29 ingo Exp $

HISTORY:    07-05-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

-  [dopplercorr](#) FOO description for function 'foo' in one line. Necessary for lookfor functionality.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SIN overloads the sin method for Analysis objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SIN overloads the sin operator for Analysis objects.
0005 %
0006 % CALL:          ao_out = sin(ao_in);
0007 %              ao_out = sin(ao_in, pl);
0008 %              ao_out = sin(ao1, pl1, ao_vector, ao_matrix, pl2);
```

```

0009 %
0010 % PARAMETERS: see help for data2D/applymethod for additional parameters
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 % >> ao.getInfo('sin')
0014 %
0015 % Get information about a specified set-plist by calling:
0016 % >> ao.getInfo('sin', 'set')
0017 %
0018 % VERSION: $Id: sin.m,v 1.23 2008/09/05 11:05:29 ingo Exp $
0019 %
0020 % HISTORY: 07-05-2007 Diepholz
0021 % Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024 %
0025 function varargout = sin(varargin)
0026
0027 % Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 % Collect input variable names
0034 in_names = cell(size(varargin));
0035 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0036
0037 % Collect all AOs
0038 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0039 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0040
0041 % Decide on a deep copy or a modify
0042 bs = copy(as, nargin);
0043
0044 % Combine plists
0045 pl = combine(pl, getDefaultPlist());
0046
0047 % Apply method to all AOs
0048 applymethod(bs, ao_invars, 'sin', pl, getDefaultPlist(), getInfo());
0049
0050 % Reshape the output to the same size of the input
0051 varargout{1} = bs;
0052 end
0053
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 % Local Functions %
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059 %
0060 % FUNCTION: getInfo
0061 %
0062 % DESCRIPTION: Get Info Object
0063 %
0064 % HISTORY: 11-07-07 M Hewitson
0065 % Creation.
0066 %
0067 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0068
0069 function ii = getInfo(varargin)
0070 if nargin == 1 && strcmpi(varargin{1}, 'None')
0071     sets = {};
0072     pl = [];
0073 else
0074     sets = {'Default'};
0075     pl = getDefaultPlist();
0076 end
0077 % Build info object
0078 ii = minfo(mfilename, 'ao', '', utils.const.categories.trig, '$Id: sin.m,v 1.23
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0079 end
0080
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %
0083 % FUNCTION: getDefaultPlist
0084 %
0085 % DESCRIPTION: Get Default Plist
0086 %
0087 % HISTORY: 11-07-07 M Hewitson
0088 % Creation.
0089 %
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091
0092 function plo = getDefaultPlist()
0093 plo = plist('axis', 'y');

```

```
0094 end
0095
0096
```

tan

PURPOSE ^

TAN overloads the tan method for Analysis objects.

SYNOPSIS ^

function varargout = tan (varargin)

DESCRIPTION ^

```
TAN overloads the tan method for Analysis objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: TAN overloads the tan operator for Analysis objects.
CALL: ao_out = tan(ao_in);
      ao_out = tan(ao_in, pl);
      ao_out = tan(ao1, pl1, ao_vector, ao_matrix, pl2);
PARAMETERS: see help for data2D/applymethod for additional parameters
The following call returns an info object for this method.
>> info = ao.getInfo('tan')
VERSION: $Id: tan.m,v 1.15 2008/09/05 11:05:29 ingo Exp $
HISTORY: 07-05-2007 Diepholz
          Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object
-  [copy](#) COPY makes a (deep) copy of the input AOs.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl_default = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % TAN overloads the tan method for Analysis objects.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: TAN overloads the tan operator for Analysis objects.
0006 %
0007 % CALL: ao_out = tan(ao_in);
0008 %        ao_out = tan(ao_in, pl);
0009 %        ao_out = tan(ao1, pl1, ao_vector, ao_matrix, pl2);
0010 %
0011 % PARAMETERS: see help for data2D/applymethod for additional parameters
0012 %
```

```

0013 % The following call returns an info object for this method.
0014 %
0015 % >> info = ao.getInfo('tan')
0016 %
0017 % VERSION: $Id: tan.m,v 1.15 2008/09/05 11:05:29 ingo Exp $
0018 %
0019 % HISTORY: 07-05-2007 Diepholz
0020 %           Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024
0025 function varargout = tan (varargin)
0026 %% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 % Collect input variable names
0033 in_names = cell(size(varargin));
0034 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036 % Collect all AOs
0037 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0038 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0039
0040 % Decide on a deep copy or a modify
0041 bs = copy(as, nargout);
0042
0043 % Combine plists
0044 pl = combine(pl, getDefaultPlist);
0045
0046 % Apply method to all AOs
0047 applymethod(bs, ao_invars, 'tan', pl, getDefaultPlist, getInfo);
0048
0049 % Reshape the ouput to the same size of the input
0050 varargout{1} = bs;
0051 end
0052
0053 %-----
0054 % Get Info Object
0055 %-----
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ao', '', utils.const.categories.trig, '$Id: tan.m,v 1.15
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0066 end
0067
0068 %-----
0069 % Get Default Plist
0070 %-----
0071 function pl_default = getDefaultPlist()
0072     pl_default = plist('axis', 'y');
0073 end
0074 % END

```


ssm

PURPOSE ^

SSM statespace model class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
SSM statespace model class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: SSM statespace model class constructor.
** This class is written using the new OO structures of MATLAB R2008a and
will not work with earlier MATLAB versions.

Public properties (read-only):

Public properties (read/write):
    name      - name of analysis object

Possible constructors:

    s = ssm()           - creates an empty statespace model
    s = ssm('a1.xml')   - creates a new statespace model by loading the
                        object from disk.
    s = ssm('a1.mat')   - creates a new statespace model by loading the
                        object from disk.
    a = ssm(plist)       - creates a statespace model from the description
                        given in the parameter list
    a = ssm(struct)     - creates a statespace model from the
                        structure returned by struct(ssm).

Parameter sets for plist constructor (in order of priority):

From XML File
-----
Construct a statespace model by loading it from an XML file.

'filename' - construct a statespace model from a filename.
Example: plist('filename', 'ss1.xml')
[default: empty string]

From MAT File
-----
Construct a statespace model by loading it from a MAT file.

'filename' - construct a statespace model from a filename.
Example: plist('filename', 'ss1.mat')
[default: empty string]
```

From Built-in System

Construct a statespace model by specifying one of the built-in system names. Built-in models are stored in directories specified by SSM_MODELS in ltpda_startup. You can list multiple directories. A built-in model m-file must follow the naming convention ssm_model_*.m.

'Buit-in' - A system name.
 Example: plist('filename', 'model_LPF_Dynamics_noparams')
 Example: plist('Built-in', '')
 will show the names of all the existing systems

'dim' - Optional parameter to reduce the model to 1, 3, or 6 dimension. [Default: empty [] - return full model]

From Repository

Construct a statespace model by retrieving it from an LTPDA repository.

'Hostname' - the repository hostname. Only those objects which are statespace models are returned.
 [default: 'localhost'];

Additional parameters:

'Database' - The database name [default: 'ltpda']
 'ID' - A vector of object IDs. [default: []]
 'CID' - Retrieve all SSM objects from a particular collection.

From a Miir

Construct a statespace model out of a MIIR object.

example : sys = ssm(themiir);

From PZ model

Construct a statespace model by retrieving it from an LTPDA repository.

example : sys=ssm(theppzmodel);

From Description

Construct a statespace model from a full description of the system.

'isADescription' >> special filed to tell the constructor which set to use

compulsory fields:

'name', 'ssnames', 'outputnames', 'inputnames', 'timestep', 'amats', ...
 'bmats', 'cmats', 'dmats', 'paramnames', 'paramvalues'

optionnals fields :

'mmats', 'amats_handles', 'bmats_handles', 'cmats_handles', 'dmats_handles'...
 'inputvarnames', 'ssvarnames', 'outputvarnames', 'paramsigmas'

example :

```
name = 'sys';
ssnames = {'ss1' 'ss2' 'ss3'};
ssvarnames = {'ssvar1' 'ssvar2' 'ssvar31' 'ssvar32'};
inputnames = {'input1' 'input2' 'input3'};
inputvarnames = {'inputvar1' 'inputvar2' 'inputvar31' 'inputvar32'};
outputnames = {'output1' 'output2' 'output3'};
outputvarnames = {'outputvar1' 'outputvar2' 'outputvar31' 'outputvar32'};
timestep = 0;
paramnames = {};
paramvalues = [];
amats = cell(3,3);
```

```
bmats = cell(3,3);
cmats = cell(3,3);
dmats = cell(3,3);
amats{1,1} = -1;
amats{2,2} = -2;
amats{3,3} = -3*eye(2);
amats{3,1} = [-1;-3];
bmats{1,1} = 1;
bmats{2,2} = 2;
bmats{3,3} = 3*eye(2);
cmats{1,1} = 1;
cmats{2,2} = 1;
cmats{3,3} = eye(2);
dmats{1,3} = [6 6];
dmats{2,1} = 6;
dmats{3,2} = [6;6];
sys = ssm(plist('isADescription', '', 'name', name, ...
'timestep', timestep, 'paramnames', paramnames, 'paramvalues', paramvalues, ...
'inputnames', inputnames, 'outputnames', outputnames, 'ssnames', ssnames, ...
'inputvarnames', inputvarnames, 'outputvarnames', outputvarnames, 'ssvarnames', ssvarnames, ...
'amats', amats, 'bmats', bmats, 'cmats', cmats, 'dmats', dmats ));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [assemble](#) assembles embedded subsystems, with exogenous inputs
-  [cell_add](#) adds corresponding matrices of same sizes or empty inside cell array
-  [cell_fusion](#) fusions a block defined matrix stored inside cell array into one matrix
-  [cell_mult](#) multiplies block defined matrix stored inside cell array
-  [cell_recut](#) cuts a matrix into blocks stored inside cell array
-  [cell_select](#) CELL_SELECT selects lines and columns of a block defined matrix stored in
-  [cellstrfind](#) cellstrfind finds matching strings inside a cellstr array
-  [char](#) CHAR convert a ssm object into a string.
-  [copy](#) COPY Make copy of ssm objects depending of the second input
-  [display](#) DISPLAY display ssm object.
-  [double](#) Convert a statespace model object to double arrays for given i/o
-  [isstable](#) tells if ssm is numerically stable
-  [kalman](#) kalman applies Kalman filtering to a discrete ssm with given i/o
-  [minreal](#) minreal gives a minimal realization of a ssm object by deleting state
-  [modifparams](#) modifparams enables to options(i_options)y and substitute parameters
-  [modiftimestep](#) modiftime modifies the timestep of a ssm object
-  [modify](#) modify allows to exectue a string to modify a ssm object
-  [reduce](#) reduce enables to do model simplification
-  [reduce_model](#) REDUCE_MODEL enables to do model simplification
-  [simulate](#) simulate simulates a discrete ssm with given inputs
-  [ssm](#) SSM statespace model class constructor.
-  [ssm2iirpz](#) ssm2iirpz converts a statespace model object to an miir or a pzmodel

-  [ssm2ss](#) SSM2SS converts a statespace model object to a MATLAB statespace object.
-  [ssmFromBuiltinSystem](#)
%%%
-  [ssmFromDescription](#)
%%%
-  [ssmFromFilename](#)
%%%
-  [ssmFromMiir](#)
%%%
-  [ssmFromPzmodel](#)
%%%
-  [ssmFromRepository](#)
%%%
-  [string](#) STRING writes a command string that can be used to recreate the input statespace model object.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version
-  [validate](#)
%%%

This function is called by:

-  [assemble](#) assembles embedded subsystems, with exogenous inputs
-  [copy](#) COPY Make copy of ssm objects depending of the second input
-  [modifitimestep](#) modifitime modifies the timestep of a ssm object
-  [ssm](#) SSM statespace model class constructor.
-  [ssmFromBuiltinSystem](#)
%%%
-  [ssmFromDescription](#)
%%%
-  [ssmFromMiir](#)
%%%
-  [ssmFromPzmodel](#)
%%%
-  [ssmFromss](#)
%%%

SUBFUNCTIONS ^

-  [function s = ssm\(varargin\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function out = SETS\(\)](#)
-  [function pl = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```

0001 % SSM statespace model class constructor.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: SSM statespace model class constructor.
0006 %
0007 % ** This class is written using the new OO structures of MATLAB R2008a and
0008 % will not work with earlier MATLAB versions.
0009 %
0010 %     Public properties (read-only):
0011 %
0012 %     Public properties (read/write):
0013 %         name         - name of analysis object
0014 %
0015 %     Possible constructors:
0016 %
0017 %         s = ssm()           - creates an empty statespace model
0018 %         s = ssm('a1.xml')   - creates a new statespace model by loading the
0019 %                               object from disk.
0020 %         s = ssm('a1.mat')   - creates a new statespace model by loading the
0021 %                               object from disk.
0022 %         a = ssm(plist)      - creates a statespace model from the description
0023 %                               given in the parameter list
0024 %         a = ssm(struct)     - creates a statespace model from the
0025 %                               structure returned by struct(ssm).
0026 %
0027 % Parameter sets for plist constructor (in order of priority):
0028 %
0029 % From XML File
0030 % -----
0031 %     Construct a statespace model by loading it from an XML file.
0032 %
0033 %     'filename' - construct a statespace model from a filename.
0034 %                  Example: plist('filename', 'ss1.xml')
0035 %                  [default: empty string]
0036 %
0037 % From MAT File
0038 % -----
0039 %     Construct a statespace model by loading it from a MAT file.
0040 %
0041 %     'filename' - construct a statespace model from a filename.
0042 %                  Example: plist('filename', 'ss1.mat')
0043 %                  [default: empty string]
0044 %
0045 % From Built-in System
0046 % -----
0047 %     Construct a statespace model by specifying one of the built-in system
0048 %     names. Built-in models are stored in directories specified by
0049 %     SSM_MODELS in ltpda_startup. You can list multiple directories. A
0050 %     built-in model m-file must follow the naming convention ssm_model_*.m.
0051 %
0052 %     'Buit-in' - A system name.
0053 %                  Example: plist('filename', 'model_LPF_Dynamics_noparams')
0054 %                  Example: plist('Built-in', '')
0055 %                  will show the names of all the existing systems
0056 %
0057 %     'dim'      - Optional parameter to reduce the model to 1, 3, or 6
0058 %                  dimension. [Default: empty [] - return full model]
0059 %
0060 % From Repository
0061 % -----
0062 %     Construct a statespace model by retrieving it from an LTPDA repository.
0063 %
0064 %     'Hostname' - the repository hostname. Only those objects which

```

```

0065 % are statespace models are returned.
0066 % [default: 'localhost'];
0067 %
0068 % Additional parameters:
0069 %
0070 % 'Database' - The database name [default: 'ltpda']
0071 % 'ID' - A vector of object IDs. [default: []]
0072 % 'CID' - Retrieve all SSM objects from a particular
0073 % collection.
0074 %
0075 % From a Miir
0076 % -----
0077 % Construct a statespace model out of a MIIR object.
0078 %
0079 % example : sys = ssm(themiir);
0080 %
0081 % From PZ model
0082 % -----
0083 % Construct a statespace model by retrieving it from an LTPDA repository.
0084 %
0085 % example : sys=ssm(thepzmodel);
0086 %
0087 %
0088 % From Description
0089 % -----
0090 % Construct a statespace model from a full description of the system.
0091 %
0092 % 'isADescription' >> special filed to tell the constructor which set to use
0093 %
0094 % compulsory fields:
0095 % 'name', 'ssnames', 'outputnames', 'inputnames', 'timestep', 'amats', ...
0096 % 'bmats', 'cmats', 'dmats', 'paramnames', 'paramvalues'
0097 %
0098 % optionnals fields :
0099 % 'mmats' 'amats_handles' 'bmats_handles' 'cmats_handles' 'dmats_handles'...
0100 % 'inputvarnames' 'ssvarnames' 'outputvarnames' 'paramsigmas'
0101 %
0102 % example :
0103 % name = 'sys';
0104 % ssnames = {'ss1' 'ss2' 'ss3'};
0105 % ssvarnames = {'ssvar1' 'ssvar2' 'ssvar31' 'ssvar32'};
0106 % inputnames = {'input1' 'input2' 'input3'};
0107 % inputvarnames = {'inputvar1' 'inputvar2' 'inputvar31' 'inputvar32'};
0108 % outputnames = {'output1' 'output2' 'output3'};
0109 % outputvarnames = {'outputvar1' 'outputvar2' 'outputvar31' 'outputvar32'};
0110 % timestep = 0;
0111 % paramnames = {};
0112 % paramvalues = [];
0113 % amats = cell(3,3);
0114 % bmats = cell(3,3);
0115 % cmats = cell(3,3);
0116 % dmats = cell(3,3);
0117 % amats{1,1} = -1;
0118 % amats{2,2} = -2;
0119 % amats{3,3} = -3*eye(2);
0120 % amats{3,1} = [-1;-3];
0121 % bmats{1,1} = 1;
0122 % bmats{2,2} = 2;
0123 % bmats{3,3} = 3*eye(2);
0124 % cmats{1,1} = 1;
0125 % cmats{2,2} = 1;
0126 % cmats{3,3} = eye(2);
0127 % dmats{1,3} = [6 6];
0128 % dmats{2,1} = 6;
0129 % dmats{3,2} = [6;6];
0130 % sys = ssm(plist('isADescription', '', 'name', name, ...
0131 % 'timestep', timestep, 'paramnames', paramnames, 'paramvalues', paramvalues, ...
0132 % 'inputnames', inputnames, 'outputnames', outputnames, 'ssnames', ssnames, ...
0133 % 'inputvarnames', inputvarnames, 'outputvarnames', outputvarnames, 'ssvarnames', ssvarnames, ...

```

```

0134 %                'amats', amats , 'bmats', bmats , 'cmats', cmats , 'dmats', dmats ));
0135 %
0136 %
0137 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0138 classdef ssm < ltpda_uoh
0139
0140 %----- Public (read/write) Properties -----
0141 properties
0142     description = '';
0143 end % End (read/write) Properties
0144
0145 %----- Private read-only Properties -----
0146 properties %(SetAccess = protected)
0147     %l linear system matrices
0148     amats      = {};
0149     mmats      = {};
0150     bmats      = {};
0151     cmats      = {};
0152     dmats      = {};
0153     isnumerical = false;
0154     timestep    = 0;
0155     % nonlinear system function handles
0156     amats_handles = {};
0157     bmats_handles = {};
0158     cmats_handles = {};
0159     dmats_handles = {};
0160     % input variables data array
0161     inputnames    = {};
0162     inputvarnames = {};
0163     inputsizes    = [];
0164     Ninputs       = 0;
0165     % state space variables data arrays
0166     ssnames       = {};
0167     ssvarnames    = {};
0168     sssizes       = [];
0169     Nss           = 0;
0170     % output variables data arrays
0171     outputnames   = {};
0172     outputvarnames = {};
0173     outputsizes   = [];
0174     Noutputs      = 0;
0175     % parameter data arrays
0176     paramnames    = {};
0177     paramvalues   = [];
0178     paramsigmas   = [];
0179     Nparams       = 0;
0180 end % End read only properties
0181
0182
0183 %----- Protected Properties -----
0184 properties (SetAccess = protected)
0185     version = '$Id: ssm.m,v 1.67 2008/09/07 18:16:13 hewitson Exp $';
0186 end
0187
0188 methods
0189
0190 end
0191
0192 %----- Public Methods -----
0193 methods(Access = public)
0194     %% Class constructor
0195     function s = ssm(varargin)
0196         import utils.const.*
0197         utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0198         % Check the supported version
0199         utils.helper.checkMatlabVersion;
0200         %% Call superclass
0201         s = s@ltpda_uoh(varargin{:}); %to inherit properties
0202

```

```

0203
0204 % Collect all ssm objects to check for copy constructor
0205 [sss, invars, rest] = utils.helper.collect_objects(varargin(:), 'ssm');
0206
0207 if isempty(rest) && ~isempty(sss)
0208     % Do copy constructor and return
0209     utils.helper.msg(msg.OPROC1, 'copy constructor');
0210     s = copy(sss, 1);
0211     for kk=1:numel(s)
0212         s(kk).addHistory(ssm.getInfo('ssm', 'None'), [], [], s(kk).hist);
0213     end
0214     return
0215 end
0216
0217 % Execute appropriate constructor
0218 switch nargin
0219     case 0 % Empty constructor
0220         s.addHistory(ssm.getInfo('ssm', 'None'), plist, {''}, []);
0221     case 1 % Plist, pzmodel, ...
0222         vin = varargin{1};
0223         if isa(vin, 'ssm') % copy constructor
0224             s = copy(vin, 1);
0225             for kk = 1:numel(vin)
0226                 utils.helper.msg(msg.PROC1, 'copying %s', vin(kk).name);
0227                 s(kk).addHistory(ssm.getInfo('ssm', 'None'), [], [], vin(kk).hist);
0228             end
0229         elseif isstruct(vin) % Struct constructor
0230             fnames = fieldnames(vin);
0231             for jj = 1:length(fnames)
0232                 f = fnames{jj};
0233                 if isstruct(vin.(f))
0234                     s.(f) = utils.helper.struct2obj(vin.(f));
0235                 else
0236                     s.(f) = vin.(f);
0237                 end
0238             end
0239         elseif isa(vin, 'plist') % constructor with a plist
0240             if isparam(vin, 'Filename')
0241                 % filename constructor
0242                 s = ssm.fromFile(vin);
0243             elseif isparam(vin, 'Built-in')
0244                 % Construct from built-in
0245                 s = ssm.ssmFromBuiltinSystem(vin);
0246             elseif isparam(vin, 'Hostname')
0247                 % Retrieve from repository
0248                 s = ssm.ssmFromRepository(vin);
0249             elseif isparam(vin, 'amats')
0250                 % construct from plist description
0251                 s = ssm.ssmFromDescription(vin);
0252             else
0253                 display('###          Unknown constructor          ###')
0254                 display('### could not find a valid parameter key ###')
0255                 display('### these are : ''Filename'', ''Built-in''          ###')
0256                 display('### these are : ''amats'', ''Hostname''          ###')
0257                 error('See above message ^^');
0258             end
0259         elseif isa(vin, 'pzmodel') % Plist constructor
0260             s = ssm.ssmFromPzmodel(vin);
0261         elseif isa(vin, 'miir') % Plist constructor
0262             s = ssm.ssmFromMiir(vin);
0263         elseif ischar(vin) % from file
0264             filename = varargin{1};
0265             s = fromFile(s, filename);
0266         else
0267             error(['### Error: ssm constructor can''t handle input of type : ', class(vin)]);
0268         end
0269         % Now validate this object
0270         s = s.validate();
0271     case 2

```



```

0272         if isa(varargin{1}, 'ssm') && isa(varargin{2}, 'plist') && isempty(varargin{2}.params)
0273             % pass to copy constructor
0274             s = ssm(varargin{1});
0275         else
0276             error('### Unknown two parameter constructor.');
```

```

0277         end
0278     otherwise
0279         [ss, invars, rest] = utils.helper.collect_objects(varargin(:), 'ssm');
0280         %% Do we have a list of SSMs as input
0281         if ~isempty(ss) && isempty(rest)
0282             s = ssm(ss);
0283         else
0284             error('### Unknown number of constructor arguments.');
```

```

0285         end
0286     end
0287 end
0288 %% copying one ssm
0289 varargout = copy(varargin)
0290 %% makes an executable string
0291 varargout = string(varargin)
0292 %% change timestep
0293 varargout = modifitimestep(varargin)
0294 %% change and substitute parameters
0295 varargout = modifparams(varargin)
0296 %% assemble systems
0297 varargout = assemble(varargin)
0298 %% transform into a pzmodel or a iir
0299 varargout = ssm2iirpz(varargin)
0300 %% transform into ABCD doubles/symb
0301 varargout = double(varargin)
0302 %% transform into a matlab ss object
0303 varargout = ssm2ss(varargin)
0304 %% model modification executing a string
0305 varargout = modify(varargin)
0306 %% simulation
0307 varargout = simulate(varargin)
0308 %% kalman filter
0309 varargout = kalman(varargin)
0310 %% model simplification (states)
0311 varargout = reduce(varargin)
0312 %% model simplification (variables)
0313 varargout = reduce_model(varargin)
0314 %% char
0315 varargout = char(varargin)
0316 %% display
0317 varargout = display(varargin)
0318 %% give minimal realization (right now needs control toolbox)
0319 varargout = minreal(varargin)
0320 %% tells if ssm is stable
0321 varargout = isstable(varargin)
0322 end
0323
0324 %----- Declaration of private methods -----
0325 methods (Access = private)
0326     %% completion and error check
0327     varargout = validate(varargin)
0328 end
0329
0330
0331 %----- Declaration of Public Static methods -----
0332 methods (Static=true, Access=public)
0333     function ii = getInfo(varargin)
0334         ii = utils.helper.generic_getInfo(varargin{:}, 'ssm');
0335     end
0336
0337     function out = VEROUT()
0338         out = '$Id: ssm.m,v 1.67 2008/09/07 18:16:13 hewitson Exp $';
0339     end
0340

```

```

0341 function out = SETS()
0342     out = {...
0343         'Default', ...
0344         'From XML File', ...
0345         'From MAT File', ...
0346         'From Built-in Model', ...
0347         'From Description', ...
0348         'From Plist', ...
0349         'From Repository', ...
0350         'From Pzmodel', ...
0351         'From Miir'};
0352 end
0353
0354 function pl = getDefaultPlist(set)
0355     sets = ssm.SETS;
0356     % Select parameter set
0357     switch set
0358     case sets{1} % Default
0359         pl = plist();
0360     case sets{2} % XML file
0361         pl = plist('filename', '');
0362     case sets{3} % MAT file
0363         pl = plist('filename', '');
0364     case sets{4} % built-in
0365         pl = plist('Built-in', '', 'dim', 3, 'withParams', true);
0366     case sets{5} % description
0367         plist('isADescription', '', 'name', 'me', 'ssnames', {'my state'}, ...
0368             'outputnames', {'force'}, 'inputnames', {'air', 'food'}, 'timestep', {0}, ...
0369             'amats', {-0.001}, 'bmats', {0, 1}, 'cmats', {1}, 'dmats', {1, 0}, ...
0370             'paramnames', {'health'}, 'paramvalues', 1, ...
0371             ... %second part, here facultative fields:
0372             'mmats', {80}, 'amats_handles', cell(1,1), 'bmats_handles', cell(1,2), ...
0373             'cmats_handles', cell(1,1), 'dmats_handles', cell(1,2), ...
0374             'inputvarnames', {'02', 'hamburger'}, ...
0375             'ssvarnames', {'energy left'}, ...
0376             'outputvarnames', {'force left to click on the mouse'}, ...
0377             'paramsigmas', 1);
0378         pl = plist();
0379     case sets{6} % plist
0380         pl = plist('plist', plist());
0381     case sets{7} % repository
0382         pl = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
0383     case sets{8} % pzmodel
0384         pl = plist();
0385     case sets{9} % miir
0386         pl = plist();
0387     otherwise
0388         error('### Unknown parameter set requested.');
```

```

0389     end
0390 end
0391
0392 %% other usefull subroutines
0393 varargout = cellstrfind(c,s,option)
0394 varargout = update_struct(varargin)
0395 %% subroutines for block defined matrix calculus
0396 a_out = cell_recut(a, rowsizes, colsizes)
0397 a_out = cell_fusion(a, rowsizes, colsizes)
0398 c = cell_mult(a,b)
0399 a = cell_add(a,b)
0400 [cell_mat_out, rowsizes_out, colsizes_out] = cell_select(varargin)
0401 end % End public static methods
0402
0403
0404 %----- Declaration of Private Static methods -----
0405 methods (Static=true, Access=private)
0406     %% ssm constructor + predefined model reduction
0407     varargout = ssm_reduced_builtIn(varargin)
0408     % retrieve from repository
0409     sys = ssmFromRepository( pl)

```

Description of ssm

```
0410     % load from file
0411     sys = ssmFromFile( pl)
0412     % create from built-in system
0413     sys = ssmFromBuiltinSystem( pli)
0414     % create from miir
0415     sys = ssmFromMiir(miirsin)
0416     % create from plist
0417     sys = ssmFromDescription(pl)
0418     % create from pzmodel
0419     sys = ssmFromPzmodel(pzmodelsin)
0420 end
0421
0422 end % End classdef
0423
0424
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

copy

PURPOSE ^

COPY Make copy of ssm objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of ssm objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of ssm objects depending of the second input

CALL:      objs = copy(objs, [1,0]);
           objs = copy(objs); Second input is assumed to be 1

VERSION:    $Id: copy.m,v 1.10 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    07-08-2008 Grynagier - made second input facultative
           11-07-2008 Diepholz
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ssm](#) SSM statespace model class constructor.

This function is called by:

-  [assemble](#) assembles embedded subsystems, with exogenous inputs
-  [kalman](#) kalman applies Kalman filtering to a discrete ssm with given i/o
-  [modifparams](#) modifparams enables to options(i_options)y and substitute parameters
-  [modiftimestep](#) modiftime modifies the timestep of a ssm object
-  [modify](#) modify allows to exectue a string to modify a ssm object
-  [reduce](#) reduce enables to do model simplification
-  [reduce_model](#) REDUCE_MODEL enables to do model simplification
-  [ssm](#) SSM statespace model class constructor.
-  [ssm2iirpz](#) ssm2iirpz converts a statespace model object to an miir or a pzmodel



%%%

SOURCE CODE ^

```
0001 % COPY Make copy of ssm objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of ssm objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %           objs = copy(objs); Second input is assumed to be 1
0008 %
0009 % VERSION:    $Id: copy.m,v 1.10 2008/09/03 16:29:33 hewitson Exp $
0010 %
0011 % HISTORY:    07-08-2008 Grynagier - made second input facultative
0012 %           11-07-2008 Diepholz
0013 %           Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 function varargout = copy(old, deepcopy)
0018
0019     %% begin function body
0020
0021     if deepcopy
0022         prov = provenance;
0023         t0 = time;
0024         ps = properties('ssm');
0025         s = size(old);
0026         obj(s(1),s(2)) = ssm;
0027         for kk=1:numel(old)
0028             utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
0029             for jj=1:length(ps)
0030                 obj(kk).(ps{jj}) = old(kk).(ps{jj});
0031             end
0032             % These should be reset on copy
0033             obj(kk).created = t0;
0034             obj(kk).creator = prov;
0035             obj(kk).version = ssm.VEROUT;
0036         end
0037     else
0038         obj = old;
0039     end
0040     varargout{1} = obj;
0041
0042 end
0043
```

char

PURPOSE ^

CHAR convert a ssm object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a ssm object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a ssm object into a string.

CALL:      string = char(ssm)

MFILE INFO: Get information about this methods by calling
             >> ssm.getInfo('char')

             Get information about a specified setplist by calling:
             >> ssm.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.9 2008/07/21 13:57:30 ingo Exp $

HISTORY:    04022007 M Hewitson
             Creation

TODO :      make it work with symbolic (wait for symb. toolbox in R2008b'?)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ssm](#) SSM statespace model class constructor.
 -  [ssmFromDescription](#)
- %%

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
- 

```
function plo = getDefaultPlist()
```

SOURCE CODE ^

```
0001 % CHAR convert a ssm object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a ssm object into a string.
0005 %
0006 % CALL:      string = char(ssm)
0007 %
0008 % MFILE INFO: Get information about this methods by calling
0009 %              >> ssm.getInfo('char')
0010 %
0011 %              Get information about a specified setplist by calling:
0012 %              >> ssm.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.9 2008/07/21 13:57:30 ingo Exp $
0015 %
0016 % HISTORY:    04022007 M Hewitson
0017 %              Creation
0018 %
0019 % TODO :      make it work with symbolic (wait for symb. toolbox in R2008b'?)
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 function varargout = char(varargin)
0024
0025 %% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 objs = utils.helper.collect_objects(varargin(:), 'ssm');
0032
0033 pstr = '';
0034
0035 for ii = 1:numel(objs)
0036     obj = objs(ii);
0037     pstr = [pstr sprintf('ssm(name=%s), ', obj.name)];
0038 end
0039
0040 pstr = strrep(pstr, '_', '\_');
0041 pstr = pstr(1:end-2);
0042
0043 varargout{1} = pstr;
0044 end
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %                               Local Functions
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 % FUNCTION:    getInfo
0053 %
0054 % DESCRIPTION: Get Info Object
0055 %
0056 % HISTORY:    110707 M Hewitson
0057 %              Creation.
0058 %
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 function ii = getInfo(varargin)
```

```
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl   = [];
0065     else
0066         sets = {'Default'};
0067         pl   = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'ssm', '', 'Output', '$Id: char.m,v 1.9 2008/07/21 13:57:30 ingo Exp $', sets, pl);
0071 end
0072
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getDefaultPlist
0076 %
0077 % DESCRIPTION:  Get Default Plist
0078 %
0079 % HISTORY:      110707 M Hewitson
0080 %               Creation.
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 function plo = getDefaultPlist()
0085     plo = plist();
0086 end
0087
0088
```


display

PURPOSE ^

DISPLAY display ssm object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY display ssm object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY display ssm object.

MFILE INFO: Get information about this methods by calling
             >> ssm.getInfo('display')

             Get information about a specified set by calling:
             >> ssm.getInfo('display', 'Default')

VERSION:     $Id: display.m,v 1.1 2008/08/06 18:35:26 ingo Exp $

HISTORY:     30012007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [isstable](#) tells if ssm is numerically stable
 -  [ssm](#) SSM statespace model class constructor.
 -  [ssmFromDescription](#)
- %%

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % DISPLAY display ssm object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY display ssm object.
0005 %
0006 % MFILE INFO: Get information about this methods by calling
0007 %             >> ssm.getInfo('display')
0008 %
0009 %             Get information about a specified set by calling:
0010 %             >> ssm.getInfo('display', 'Default')
0011 %
0012 % VERSION:      $Id: display.m,v 1.1 2008/08/06 18:35:26 ingo Exp $
0013 %
0014 % HISTORY:      30012007 Diepholz
0015 %                Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 function varargout = display(varargin)
0020 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0021 %% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026 objs = utils.helper.collect_objects(varargin(:), 'ssm');
0027 txt = utils.helper.objdisp(objs);
0028 if nargin == 0
0029     for ii=1:length(txt)
0030         disp(txt{ii});
0031     end
0032 else
0033     varargout{1} = txt;
0034 end
0035 end
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037 % Local Functions %
0038 %
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040 function ii = getInfo(varargin)
0041     if nargin == 1 && strcmpi(varargin{1}, 'None')
0042         sets = {};
0043         pl = [];
0044     else
0045         sets = {'Default'};
0046         pl = getDefaultPlist;
0047     end
0048     % Build info object
0049     ii = minfo(mfilename, 'ssm', '', 'Output', '$Id: display.m,v 1.1 2008/08/06 18:35:26 ingo Exp $', sets, pl);
0050 end
0051
0052 function plo = getDefaultPlist()
0053     plo = plist();
0054 end
0055
0056
0057
```

isstable

PURPOSE ^

tells if ssm is numerically stable

SYNOPSIS ^

function varargout = isstable(varargin)

DESCRIPTION ^

```
tells if ssm is numerically stable
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: ISSTABLE tells if ssm is numerically stable

MFILE INFO: Get information about this methods by calling
>> ssm.getInfo('display')

            Get information about a specified set by calling:
>> ssm.getInfo('display', 'Default')

VERSION:    $Id: display.m,v 1.1 2008/08/06 18:35:26 ingo Exp $

HISTORY:    30012007 Diepholz
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [display](#) DISPLAY display ssm object.
-  [double](#) Convert a statespace model object to double arrays for given i/o

This function is called by:

-  [ssm](#) SSM statespace model class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % tells if ssm is numerically stable
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ISSTABLE tells if ssm is numerically stable
0005 %
0006 % MFILE INFO: Get information about this methods by calling
0007 % >> ssm.getInfo('display')
0008 %
0009 %           Get information about a specified set by calling:
0010 % >> ssm.getInfo('display', 'Default')
0011 %
0012 % VERSION:    $Id: display.m,v 1.1 2008/08/06 18:35:26 ingo Exp $
0013 %
0014 % HISTORY:    30012007 Diepholz
```

```

0015 %                               Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 function varargout = isstable(varargin)
0020 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0021 %%% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026 objs = utils.helper.collect_objects(varargin(:), 'ssm');
0027
0028 for i=1:length(objs)
0029     [A B C D Ts] = double(objs(i), plist('inputs','ALL','states',
'ALL','outputs','ALL'));
0030     if Ts == 0
0031         vp = eig(A);
0032         ninst = length(find(real(vp)>0));
0033         isst(i) = ninst==0;
0034     else
0035         vp = eig(A);
0036         ninst = length(find(abs(vp)>1));
0037         isst(i) = ninst==0;
0038     end
0039 end
0040 if sum(isst)<length(isst)
0041     display('warning, at least one system is not stable :')
0042     for i=1:length(objs)
0043         if ~isst(i)
0044             display(objs(i).name)
0045             display(objs(i).description)
0046         end
0047     end
0048 end
0049 varargout = {isst};
0050 end
0051
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053 %                               Local Functions
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ssm', '', 'Output', '$Id: display.m,v 1.1 2008/08/06 18:35:26
ingo Exp $', sets, pl);
0066 end
0067
0068 function plo = getDefaultPlist()
0069     plo = plist();
0070 end
0071
0072

```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input statespace model object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input statespace model
object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate the
input statespace model object.

CALL:      cmd = string(ssm)

INPUTS:    cmd - statespace model object

OUTPUTS:    ssm - command string to create the statespace model object

M-FILE INFO: Get information about this methods by calling
              >> ssm.getInfo('string')

              Get information about a specified set-plist by calling:
              >> ssm.getInfo('string', 'None')

VERSION:    $Id: string.m,v 1.8 2008/07/21 13:57:30 ingo Exp $

HISTORY:    29-03-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input statespace model object.

This function is called by:

-  [ssm](#) SSM statespace model class constructor.
-  [string](#) STRING writes a command string that can be used to recreate the input statespace model object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input statespace
```

```

model object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input statespace model object.
0006 %
0007 % CALL:         cmd = string(ssm)
0008 %
0009 % INPUTS:       cmd - statespace model object
0010 %
0011 % OUTPUTS:      ssm - command string to create the statespace model object
0012 %
0013 % M-FILE INFO:  Get information about this methods by calling
0014 %               >> ssm.getInfo('string')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> ssm.getInfo('string', 'None')
0018 %
0019 % VERSION:      $Id: string.m,v 1.8 2008/07/21 13:57:30 ingo Exp $
0020 %
0021 % HISTORY:      29-03-2007 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = string(varargin)
0027 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 %%% Get minfo objects
0036 objs = utils.helper.collect_objects(varargin(:), 'ssm');
0037
0038 cmd = '';
0039 for kk = 1:numel(objs)
0040     pl = objs(kk).hist.plistUsed;
0041     cmd = [cmd 'ssm(' string(pl) '), '];
0042 end
0043
0044 cmd = cmd(1:end-2);
0045
0046 if numel(objs) > 1
0047     cmd = ['[' cmd ']'];
0048 end
0049
0050 varargout{1} = cmd;
0051 end
0052
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054 %                               Local Functions                               %
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 % FUNCTION:      getInfo
0060 %
0061 % DESCRIPTION:   Get Info Object
0062 %
0063 % HISTORY:      11-07-07 M Hewitson
0064 %               Creation.
0065 %
0066 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0067
0068 function ii = getInfo(varargin)
0069 if nargin == 1 && strcmpi(varargin{1}, 'None')
0070     sets = {};
0071     pl = [];
0072 else
0073     sets = {'Default'};
0074     pl = getDefaultPlist;
0075 end
0076 % Build info object
0077 ii = minfo(mfilename, 'ssm', '', 'Output', '$Id: string.m,v 1.8 2008/07/21 13:57:30
ingo Exp $', sets, pl);
0078 end
0079
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 % FUNCTION:      getDefaultPlist
0083 %
0084 % DESCRIPTION:   Get Default Plist
0085 %

```

```
0086 % HISTORY:      11-07-07 M Hewitson
0087 %
0088 %
0089 %%%%%%%%%%%
0090
0091 function plo = getDefaultPlist()
0092     plo = plist();
0093 end
0094
```

assemble

PURPOSE ^

assembles embedded subsystems, with exogenous inputs

SYNOPSIS ^

function varargout = assemble(varargin)

DESCRIPTION ^

```
assembles embedded subsystems, with exogenous inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: assemble assembles embedded subsystems, with exogenous inputs
CALL: [sys] = ltpda_ss_assemble(sys_array)
INPUTS: sys_array - array of systems to assemble
OUTPUTS: sys - assembled system

***** There are no parameters *****

VERSION: '$Id: ltpda_ss_assemble.m,v 1.2 2008/02/20 09:20:07 adrien Exp$'
M-FILE INFO: Get information about this methods by calling
              >> ssm.getInfo('assemble')

              Get information about a specified set-plist by calling:
              >> ssm.getInfo('assemble', 'Default')

HISTORY:
09-07-2008 A Grynagier
29-05-2008 A Grynagier
17-04-2008 A Grynagier
23-01-2008 A Grynagier
29-01-2008 A Grynagier

TO DO :
needs work on D inversion.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of ssm objects depending of the second input
-  [ssm](#) SSM statespace model class constructor.
-  [validate](#)

This function is called by:

 `ssm` SSM statespace model class constructor.

SUBFUNCTIONS ^

-  `function ii = getInfo(varargin)`
-  `function out = getDefaultPlist(set)`

SOURCE CODE ^

```
0001 % assembles embedded subsystems, with exogenous inputs
0002 %%%%%%%%%%
0003 %
0004 % DESCRIPTION: assemble assembles embedded subsystems, with exogenous inputs
0005 %
0006 % CALL: [sys] = ltpda_ss_assemble(sys_array)
0007 %
0008 % INPUTS: sys_array - array of systems to assemble
0009 %
0010 % OUTPUTS: sys - assembled system
0011 %
0012 % ***** There are no parameters *****
0013 %
0014 % VERSION: '$Id: ltpda_ss_assemble.m,v 1.2 2008/02/20 09:20:07 adrien Exp$'
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 %               >> ssm.getInfo('assemble')
0018 %
0019 %               Get information about a specified set-plist by calling:
0020 %               >> ssm.getInfo('assemble', 'Default')
0021 %
0022 % HISTORY:
0023 % 09-07-2008 A Grynagier
0024 % 29-05-2008 A Grynagier
0025 % 17-04-2008 A Grynagier
0026 % 23-01-2008 A Grynagier
0027 % 29-01-2008 A Grynagier
0028 %
0029 % TO DO :
0030 % needs work on D inversion.
0031 %%%%%%%%%%
0032
0033 function    varargout = assemble( varargin )
0034
0035 % starting initial checks
0036 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0037
0038 % Check if this is a call for parameters
0039 if utils.helper.isinfocall(varargin{:})
0040     varargout{1} = getInfo(varargin{3});
0041     return
0042 end
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
```

```

0048 % Collect all SSMs and plists
0049 [sys, ssm_invars] = utils.helper.collect_objects(varargin(:), 'ssm', in_names);
0050
0051 Nsys = numel(sys);
0052
0053 if Nsys < 2
0054     error('### Two or more systems are needed to assemble.');
```

0055 end

```

0056
0057 % We want to force a copy - this is not a modify method
0058 if nargin < 1
0059     error('### assemble cannot be used as a modifier. Please give an output variable.');
```

0060 end

```

0061
0062 % Decide on a deep copy or a modify, depending on the output
0063 sys_array = copy(sys, nargin);
0064
0065 %% begin function body
0066 sys_out = ssm;
```

0067

```

0068 %% Putting together parameter fields
0069 for i=1:Nsys
0070     sys_out.paramnames = [sys_out.paramnames sys_array(i).paramnames]; %ok<AGROW>
0071     sys_out.paramvalues = [sys_out.paramvalues sys_array(i).paramvalues]; %ok<AGROW>
0072     sys_out.paramsigmas = [sys_out.paramsigmas sys_array(i).paramsigmas]; %ok<AGROW>
0073 end
0074 sys_out.isnumerical = true;
0075 for i=1:Nsys
0076     sys_out.isnumerical = sys_out.isnumerical*sys_array(i).isnumerical;
```

0077 end

```

0078
0079 %% merging ss data : ss* output* *input
0080
0081 names = cell(1,0);
0082 ssnames = cell(1,0);
0083 ssvarnames = cell(1,0);
0084 sssizes = zeros(1,0);
0085 ssposition = zeros(2,0);
0086 outputnames = cell(1,0);
0087 outputvarnames = cell(1,0);
0088 outputsizes = zeros(1,0);
0089 outputposition = zeros(2,0);
0090 inputnames = cell(1,0);
0091 inputvarnames = cell(1,0);
0092 inputsizes = zeros(1,0);
0093 inputposition = zeros(2,0);
0094 for i=1:Nsys
0095     names = [names sys_array(i).name];
0096     ssnames = [ssnames sys_array(i).ssnames];
0097     ssvarnames = [ssvarnames sys_array(i).ssvarnames];
0098     sssizes = [sssizes sys_array(i).sssizes];
0099     ssposition = [ssposition [i*ones(1,sys_array(i).Nss) ; 1:sys_array(i).Nss] ];
0100     outputnames = [outputnames sys_array(i).outputnames];
0101     outputvarnames = [outputvarnames sys_array(i).outputvarnames];
0102     outputsizes = [outputsizes sys_array(i).outputsizes];
0103     outputposition = [outputposition [i*ones(1,sys_array(i).Noutputs) ; 1:sys_array(i).Noutputs] ];
0104     inputvarnames = [inputvarnames sys_array(i).inputvarnames];
0105     inputnames = [inputnames sys_array(i).inputnames];
0106     inputsizes = [inputsizes sys_array(i).inputsizes];
0107     inputposition = [inputposition [i*ones(1,sys_array(i).Ninputs) ; 1:sys_array(i).Ninputs] ];
0108 end
0109 %% data already good to store
0110 sys_out.ssnames = ssnames;
0111 sys_out.ssvarnames = ssvarnames;
0112 sys_out.outputnames = outputnames;
0113 sys_out.outputvarnames = outputvarnames;
0114
0115 %% non redundancy checks
0116 for i=1:length(names)
```

```

0117 [pos_in, sum_in] = ssm.cellstrfind(names, names{i}, 'all');
0118 if sum_in>1
0119     error([num2str(sum_in), ' systems are named ', names{i}])
0120 end
0121 end
0122 for i=1:length(ssnames)
0123     [pos_in, sum_in] = ssm.cellstrfind(ssnames, ssnames{i}, 'all');
0124     if sum_in>1
0125         error([num2str(sum_in), ' states are named ', ssnames{i}])
0126     end
0127 end
0128 for i=1:length(outputnames)
0129     [pos_in, sum_in] = ssm.cellstrfind(outputnames, outputnames{i}, 'all');
0130     if sum_in>1
0131         error([num2str(sum_in), ' outputs are named ', outputnames{i}])
0132     end
0133 end
0134
0135 %% fields for outer inputs
0136 inputisouter = false(1, length(outputnames));
0137 for i=1:length(inputnames)
0138     [pos_in, sum_in] = ssm.cellstrfind(outputnames, inputnames(i), 'all');
0139     if sum_in==0
0140         inputisouter(i)=true;
0141     end
0142 end
0143
0144 %% building A, M and C matrices
0145 amats = {};
0146 mmats = {};
0147 cmats = {};
0148 for i=1:Nsys
0149     amats = ...
0150         [amats
0151          cell(size(sys_array(i).amats,1),size(sys_array(i).amats,2))
0152          sys_array(i).amats ] ; ...
0153     mmats = ...
0154         [mmats
0155          cell(size(sys_array(i).mmats,1),size(mmats,2))
0156          sys_array(i).mmats ] ; ...
0157     cmats = ...
0158         [cmats
0159          cell(size(sys_array(i).cmats,1),size(cmats,2))
0160          sys_array(i).cmats ] ; ...
0161 end
0162 sys_out.mmats = mmats;
0163
0164 %% building B_in and D_in matrices
0165 B_in = cell( length(ssnames), length(outputnames) );
0166 D_in = cell( length(outputnames), length(outputnames) );
0167 for i=1:length(outputnames)
0168     [pos_in, sum_in] = ssm.cellstrfind(inputnames, outputnames{i}, 'all');
0169     if sum_in>0
0170         for pos = pos_in
0171             systemNumber = inputposition(1,pos);
0172             inputNumber = inputposition(2,pos);
0173             B_in(ssposition(1,:)==systemNumber, i) = sys_array(systemNumber).bmats(:,inputNumber);
0174             D_in(outputposition(1,:)==systemNumber, i) = sys_array(systemNumber).dmats(:,inputNumber);
0175         end
0176     end
0177 end
0178
0179 %% building B_ext and D_ext matrices
0180 B_ext = cell( length(ssnames), length(inputnames) );
0181 D_ext = cell( length(outputnames), length(inputnames) );
0182 pos_keep = false(1, length(inputnames));
0183 for i=1:length(inputnames)
0184     [pos_in, sum_in] = ssm.cellstrfind(outputnames, inputnames(i), 'all');
0185     if sum_in==0
0186         [pos_in, sum_in] = ssm.cellstrfind(inputnames, inputnames(i), 'all');
0187         for pos = pos_in;
0188             systemNumber = inputposition(1,pos);

```

```

0186         inputNumber = inputposition(2,pos);
0187         B_ext( ssposition(1,:)==systemNumber, pos_in(1) ) = sys_array(systemNumber).bmats(:,inputNumber);
0188         D_ext( outputposition(1,:)==systemNumber, pos_in(1) ) = sys_array(systemNumber).dmats(:,inputNumber);
0189         pos_keep(pos_in(1)) = true;
0190     end
0191 end
0192 end
0193
0194 %% getting amats, bmats, cmats, dmats
0195 B_ext = B_ext(:,pos_keep);
0196 D_ext = D_ext(:,pos_keep);
0197 sys_out.inputnames = inputnames(pos_keep);
0198 sys_out.inputvarnames = inputvarnames(pos_keep);
0199
0200 %% flattening and getting constant feedthrough
0201 D_in_f = ssm.cell_fusion(D_in, outputsizes, outputsizes);
0202 id_D_inv = (eye(size(D_in_f)) - D_in_f)^-1;
0203 id_D_inv = ssm.cell_recut(id_D_inv, outputsizes, outputsizes);
0204 sys_out.amats = ssm.cell_add(amats, ssm.cell_mult(B_in, ssm.cell_mult(id_D_inv, cmats)));
0205 sys_out.bmats = ssm.cell_add(B_ext, ssm.cell_mult(B_in, ssm.cell_mult(id_D_inv, D_ext)));
0206 sys_out.cmats = ssm.cell_mult(id_D_inv, cmats);
0207 sys_out.dmats = ssm.cell_mult(id_D_inv, D_ext);
0208
0209 %% building new name and strings
0210 name = 'assembled( ';
0211 for i = 1:Nsys
0212     if i==1
0213         name = [name, sys_array(i).name ];
0214     elseif i<Nsys
0215         name = [name, ' + ', sys_array(i).name];
0216     else
0217         name = [name, ' + ', sys_array(i).name, ')'];
0218     end
0219 end
0220 name = [name ')'];
0221 sys_out.name = name;
0222
0223 %% getting timestep and checking consistency
0224 for i=1:Nsys
0225     if i == 1
0226         sys_out.timestep = sys_array(i).timestep;
0227     elseif ~ sys_out.timestep == sys_array(i).timestep
0228         error(['error because systems 1 and ', num2str(i), ...
0229             ' named ', sys_array(i).name, ' and ', sys_array(i).name, ...
0230             ' have different timesteps :', ...
0231             num2str(sys_array(i).timestep), ' and ', num2str(sys_array(i).timestep) ]);
0232     end
0233 end
0234
0235 %% setting history
0236 h = history();
0237 for i=1:Nsys
0238     h(i) = sys_array(i).hist;
0239 end
0240 sys_out.addHistory(ssm.getInfo(mfilename), plist , {''}, h );
0241 %% parsing output
0242 validate(sys_out);
0243 varargout = {sys_out};
0244 end
0245
0246
0247 %-----
0248 % Get Info Object
0249 %-----
0250 function ii = getInfo(varargin)
0251     if nargin == 1 && strcmpi(varargin{1}, 'None')
0252         sets = {};
0253         pls = [];
0254     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})

```

```
0255     sets{1} = varargin{1};
0256     pls = getDefaultPlist(sets{1});
0257 else
0258     sets = {'Default'};
0259     pls = [];
0260     for kk=1:numel(sets)
0261         pls = [pls getDefaultPlist(sets{kk})];
0262     end
0263 end
0264 % Build info object
0265 ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
0266 end
0267
0268 %-----
0269 % Get Default Plist
0270 %-----
0271 function out = getDefaultPlist(set)
0272     switch set
0273     case 'Default'
0274         out = plist;
0275     otherwise
0276         error('### Unknown set [%s] to get the default plist.', set);
0277     end
0278 end
```

double

PURPOSE ^

Convert a statespace model object to double arrays for given i/o

SYNOPSIS ^

function varargout = double(varargin)

DESCRIPTION ^

```
Convert a statespace model object to double arrays for given i/o
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: double converts a statespace model object to double arrays.

CALL:          [A B C D Ts inputvarnames2 ssvarnames2 outputvarnames2] = double(ssm, options);

INPUTS :
    ssm        - a ssm object
    options    - an options plist

OPTIONS :
plist with parameters 'inputs', 'states' and 'outputs' to indicate which
inputs, states and outputs variables are taken in account. This requires
proper variable naming. If a variable called appears more that once it
will be used once only.
The field may be :
    - a cellstr containing the resp. input/state/output *variable* names
    - a logical indexing the resp. input/state/output *variables*
      names. Index is stored in a cell array, each cell
      correponding to one input/state/output block.
    - a double indexing the resp. input/state/output *variables*
      names. Index is stored in a cell array, each cell
      correponding to one input/state/output block.
    - 'ALL', this string indicates all i/o variables will be
      given

OUTPUTS :
    A          - the As matrices in one double array
    B          - the Bs matrices in one double array
    C          - the Cs matrices in one double array
    D          - the Ds matrices in one double array
    Ts         - the sampling time of the system. 0 is
                  continuous
    inputvarnames2 - the variable names corresponding to the
                  columns of B and D
    ssvarnames2  - the variable names corresponding to the
                  rows of A and B, cols. of A and C
    outputvarnames2 - the variable names corresponding to the
                  rows of C and D

VERSION:      $Id: double.m,v 1.6 2008/06/08 18:44:07 adrien Exp $

M-FILE INFO:  Get information about this methods by calling
                >> ssm.getInfo('double')

                Get information about a specified set-plist by calling:
                >> ssm.getInfo('double', 'Default')

HISTORY:      07-08-2008 Grynagier
                16-04-08 M Hewitson
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 **reduce_model** REDUCE_MODEL enables to do model simplification

This function is called by:

-  **cell_recut** cuts a matrix into blocks stored inside cell array
-  **isstable** tells if ssm is numerically stable
-  **kalman** kalman applies Kalman filtering to a discrete ssm with given i/o
-  **ssm** SSM statespace model class constructor.
-  **ssm2iirpz** ssm2iirpz converts a statespace model object to an miir or a pzmodel
-  **ssm2ss** SSM2SS converts a statespace model object to a MATLAB statespace object.

SUBFUNCTIONS [^]

-  **function ii = getInfo(varargin)**
-  **function plo = getDefaultPlist(set)**

SOURCE CODE [^]

```

0001 % Convert a statespace model object to double arrays for given i/o
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: double converts a statespace model object to double arrays.
0005 %
0006 % CALL:          [A B C D Ts inputvarnames2 ssvarnames2 outputvarnames2] = double(ssm,
options);
0007 %
0008 % INPUTS :
0009 %         ssm      - a ssm object
0010 %         options  - an options plist
0011 %
0012 % OPTIONS :
0013 % plist with parameters 'inputs', 'states' and 'outputs' to indicate which
0014 % inputs, states and outputs variables are taken in account. This requires
0015 % proper variable naming. If a variable called appears more that once it
0016 % will be used once only.
0017 % The field may be :
0018 %         - a cellstr containing the resp. input/state/output *variable* names
0019 %         - a logical indexing the resp. input/state/output *variables*
0020 %         names. Index is stored in a cell array, each cell
0021 %         corresponding to one input/state/output block.
0022 %         - a double indexing the resp. input/state/output *variables*
0023 %         names. Index is stored in a cell array, each cell
0024 %         corresponding to one input/state/output block.
0025 %         - 'ALL', this string indicates all i/o variables will be
0026 %         given
0027 %
0028 %
0029 % OUTPUTS :   A           - the As matrices in one double array
0030 %             B           - the Bs matrices in one double array
0031 %             C           - the Cs matrices in one double array
0032 %             D           - the Ds matrices in one double array
0033 %             Ts          - the sampling time of the system. 0 is
0034 %             continuous
0035 %             inputvarnames2 - the variable names corresponding to the
0036 %             columns of B and D
0037 %             ssvarnames2  - the variable names corresponding to the
0038 %             rows of A and B, cols. of A and C
0039 %             outputvarnames2 - the variable names corresponding to the
0040 %             rows of C and D
0041 %
0042 %
0043 % VERSION:     $Id: double.m,v 1.6 2008/06/08 18:44:07 adrien Exp $
0044 %
0045 % M-FILE INFO: Get information about this methods by calling
0046 %             >> ssm.getInfo('double')
0047 %
0048 %             Get information about a specified set-plist by calling:
0049 %             >> ssm.getInfo('double', 'Default')
0050 %
0051 % HISTORY:     07-08-2008 Grynagier
0052 %             16-04-08 M Hewitson
0053 %             Creation
0054 %
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```



```

0056
0057 function varargout = double(varargin)
0058
0059 %% starting initial checks
0060 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0061
0062 % Check if this is a call for parameters
0063 if utils.helper.isinfocall(varargin{:})
0064     varargout{1} = getInfo(varargin{3});
0065     return
0066 end
0067
0068 % checking number of inputs work data
0069 if ~nargin==2, error('wrong number of inputs!'), end
0070
0071 if ~isequal(class(varargin{1}), 'ssm'), error(['argument is not a ssm but a ',
class(varargin{1})]), end
0072 if ~isequal(class(varargin{2}), 'plist'), error(['argument is not a plist but a ',
class(varargin{2})]), end
0073
0074 sys = varargin{1};
0075 options = combine(varargin{2}, getDefaultPlist('Default'));
0076 %% begin function body
0077 %making model simplifications
0078 sys = reduce_model(sys, options);
0079
0080 % Convert to double arrays
0081
0082 M = ssm.cell_fusion(sys.mmats, sys.sssizes, sys.sssizes);
0083 A = (M^-1) * ssm.cell_fusion(sys.amats, sys.sssizes, sys.sssizes);
0084 B = (M^-1) * ssm.cell_fusion(sys.bmats, sys.sssizes, sys.inputsizes);
0085 C = ssm.cell_fusion(sys.cmats, sys.outputsizes, sys.sssizes);
0086 D = ssm.cell_fusion(sys.dmats, sys.outputsizes, sys.inputsizes);
0087 Ts = sys.timestep;
0088
0089 inputvarnames3 = cell(1, sum(sys.inputsizes));
0090 k=1;
0091 for i=1:sys.Ninputs
0092     for j=1:sys.inputsizes(i)
0093         inputvarnames3{k} = sys.inputvarnames{i}{j};
0094         k = k+1;
0095     end
0096 end
0097 ssvarnames3 = cell(1, sum(sys.sssizes));
0098 k=1;
0099 for i=1:sys.Nss
0100     for j=1:sys.sssizes(i)
0101         ssvarnames3{k} = sys.ssvarnames{i}{j};
0102         k = k+1;
0103     end
0104 end
0105 outputvarnames3 = cell(1, sum(sys.outputsizes));
0106 k=1;
0107 for i=1:sys.Noutputs
0108     for j=1:sys.outputsizes(i)
0109         outputvarnames3{k} = sys.outputvarnames{i}{j};
0110         k = k+1;
0111     end
0112 end
0113
0114 %% parsing output
0115 varargout = {A B C D Ts inputvarnames3 ssvarnames3 outputvarnames3};
0116
0117 end
0118
0119
0120
0121 function ii = getInfo(varargin)
0122 if nargin == 1 && strcmpi(varargin{1}, 'None')
0123     sets = {};
0124     pls = [];
0125 elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0126     sets{1} = varargin{1};
0127     pls = getDefaultPlist(sets{1});
0128 else
0129     sets = {'Default'};
0130     pls = [];
0131     for kk=1:numel(sets)
0132         pls = [pls getDefaultPlist(sets{kk})];
0133     end
0134 end
0135 % Build info object
0136 ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38
ingo Exp $', sets, pls);
0137 end
0138

```



```
0139 function plo = getDefaultPlist(set)
0140     switch set
0141     case 'Default'
0142         plo = ssm.getInfo('reduce_model', 'Default').plists;
0143     otherwise
0144         error('### Unknown parameter set [%s].', set);
0145     end
0146 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

kalman

PURPOSE ^

kalman applies Kalman filtering to a discrete ssm with given i/o

SYNOPSIS ^

function varargout = kalman(varargin)

DESCRIPTION ^

kalman applies Kalman filtering to a discrete ssm with given i/o
%%%

DESCRIPTION: %kalman applies Kalman filtering to a discrete ssm with given i/o

CALL: [ao_out, plist_data_out] = ...
kalman(sys, plist_inputs_noise, plist_inputs_aos, plist_inputs_constants)

- INPUTS:
- sys, (array of) ssm object
 - plist_inputs_noise contains 2 fields :
 - 'input variable names' with a cell vector of strings of the desired input variable names
 - 'covariance' which gives the covariance of this noise between the different corresponding inputs given for the *TIME CONTINUOUS* noise model
 - plist_inputs_aos contains 2 fields :
 - 'input variable names' with a cell vector of strings of the desired input variable names
 - 'mean' which gives the aos for the different corresponding inputs
 - plist_inputs_constants contains 2 fields :
 - 'input variable names' with a cell vector of strings of the desired input variable names
 - 'constants' which gives the DC value for the different corresponding inputs
 - 2 arrays telling which blocks should be stored and returned
 - 'return states' : for blocks of the state space (ex [8])
 - 'return outputs' : for blocks of the output (ex [1 5])
 - 1 field 'ssini' is a cell array of vectors that give the initial position for estimation. 0 is the default.

- OUTPUTS:
- ao_out
(array of) output analysis object containing the output aos
 - plist_data_out contains 3 fields
 - 'K' the Kalman feedback
 - 'P' the state covariance
 - 'Z' the output covariance

Main assumptions : noise is stationnary
system is invariant
noise is white
process and measurement noises may be correlated correlated
Right now the algrithm requires the control toolbox

M-FILE INFO: Get information about this methods by calling
>> ssm.getInfo('simulate')

Get information about a specified set-plist by calling:
>> ssm.getInfo('simulate', 'Default')

***** THERE ARE NO DEFAULT PARAMETERS *****

VERSION: \$Id:\$

HISTORY:
10-08-2008 a. Grynagier - revival
%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of ssm objects depending of the second input
-  [double](#) Convert a statespace model object to double arrays for given i/o

This function is called by:

-  [ssm](#) SSM statespace model class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 %kalman applies Kalman filtering to a discrete ssm with given i/o
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: %kalman applies Kalman filtering to a discrete ssm with
0005 % given i/o
0006 %
0007 % CALL: [ao_out, plist_data_out] = ...
0008 % kalman(sys, plist_inputs_noise, plist_inputs_aos, plist_inputs_constants)
0009 %
0010 % INPUTS:
0011 %     - sys, (array of) ssm object
0012 %     - plist_inputs_noise contains 2 fields :
0013 %         - 'input variable names' with a cell vector of strings of
0014 %           the desired input variable names
0015 %         - 'covariance' which gives the covariance of this noise between
0016 %           the different corresponding inputs given for the *TIME
0017 %           CONTINUOUS* noise model
0018 %     - plist_inputs_aos contains 2 fields :
0019 %         - 'input variable names' with a cell vector of strings of
0020 %           the desired input variable names
0021 %         - 'mean' which gives the aos for the different
0022 %           corresponding inputs
0023 %     - plist_inputs_constants contains 2 fields :
0024 %         - 'input variable names' with a cell vector of strings of
0025 %           the desired input variable names
0026 %         - 'constants' which gives the DC value for the different
0027 %           corresponding inputs
0028 %     - 2 arrays telling which blocks should be stored and returned
0029 %         - 'return states' : for blocks of the state space (ex [8])
0030 %         - 'return outputs' : for blocks of the output (ex [1 5])
0031 %     - 1 field 'ssini' is a cell array of vectors that give the
0032 %       initial position for estimation. 0 is the default.
0033 %
0034 % OUTPUTS:
0035 %     -ao_out
0036 %       (array of) output analysis object containing the output
0037 %       aos
0038 %     -plist_data_out contains 3 fields
0039 %         - 'K' the Kalman feedback
0040 %         - 'P' the state covariance
0041 %         - 'Z' the output covariance
0042 %
0043 % Main assumptions : noise is stationnary
0044 %                   system is invariant
0045 %                   noise is white
0046 %                   process and measurement noises may be correlated correlated
0047 %                   Right now the algrithm requires the control toolbox
0048 %
0049 % M-FILE INFO: Get information about this methods by calling
0050 %               >> ssm.getInfo('simulate')
0051 %
0052 %               Get information about a specified set-plist by calling:
0053 %               >> ssm.getInfo('simulate', 'Default')
0054 %
0055 % ***** THERE ARE NO DEFAULT PARAMETERS *****
0056 %
0057 % VERSION: $Id:$
```

```

0058 %
0059 % HISTORY:
0060 % 10-08-2008 a. Grynagier - revival
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062
0063 function varargout = kalman(varargin)
0064
0065 %% starting initial checks
0066 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0067
0068 % Check if this is a call for parameters
0069 if utils.helper.isinfocall(varargin{:})
0070     varargout{1} = getInfo(varargin{3});
0071     return
0072 end
0073
0074 % checking number of inputs work data
0075 if ~nargin==2, error('wrong number of inputs!'), end
0076
0077 if ~isequal(class(varargin{1}), 'ssm'), error(['argument is not a ssm but a ',
class(varargin{1})]), end
0078 if ~isequal(class(varargin{2}), 'plist'), error(['argument is not a plist but a ',
class(varargin{2})]), end
0079
0080 sys = varargin{1};
0081 plist_inputs = varargin{2};
0082 %% begin function body
0083
0084 i_sys=1;
0085 % for i_sys = 1:Nsys
0086 %% retrieve system infos
0087 if ~sys(i_sys).isnumerical
0088     error(['error because system ', sys(i_sys).name, ' is not numerical']);
0089 end
0090 M2 = ssm.cell_fusion(sys(i_sys).mmats, sys(i_sys).sssizes, sys(i_sys).sssizes);
0091 Minv = ssm.cell_recut(M2^-1, sys(i_sys).sssizes, sys(i_sys).sssizes);
0092 if ~isequal(M2, eye(size(M2)))
0093     utils.helper.msg(utils.const.msg.MNAME, 'warning, M is not the identity!!! It is:');
0094 end
0095 sssizes = sys(i_sys).sssizes;
0096 timestep = sys(i_sys).timestep;
0097 Ninputs = sys(i_sys).Ninputs;
0098 Noutputs = sys(i_sys).Noutputs;
0099 Nss = sys(i_sys).Nss;
0100 outputsizes = sys(i_sys).outputsizes;
0101 inputsizes = sys(i_sys).inputsizes;
0102
0103 ssini = find(plist_inputs, 'ssini');
0104 if isempty(ssini)
0105     ssini = cell(sys(i_sys).Nss,1);
0106     for i=1:sys(i_sys).Nss
0107         ssini{i} = zeros(sys(i_sys).sssizes(i),1);
0108     end
0109 end
0110
0111 %% collecting and finding position of data **for inputs** option plists
0112 noise_varnames = find(plist_inputs, 'noise variable names');
0113 Nnoise_in = length(noise_varnames);
0114 noise_in = find(plist_inputs, 'variance');
0115 noise_pos = zeros(Nnoise_in,2);
0116
0117 aos_varnames = find(plist_inputs, 'aos variable names');
0118 Naos_in = length(aos_varnames);
0119 aos_in = find(plist_inputs, 'aos');
0120 aos_pos = zeros(size(aos_varnames),2);
0121
0122 constants_varnames = find(plist_inputs, 'constant variable names');
0123 Nconstants_in = length(constants_varnames);
0124 constants_in = find(plist_inputs, 'constants');
0125 constants_pos = zeros(size(constants_varnames),2);
0126
0127 outputs_varnames = find(plist_inputs, 'output variable names');
0128 Noutputs_in = length(outputs_varnames);
0129 outputs_in = find(plist_inputs, 'outputs');
0130 outputs_pos = zeros(size(outputs_varnames),1);
0131
0132
0133
0134 %% getting A - B - C - D - E - F - H realization
0135 % A - C are the state matrix
0136 % B - D are the signal matrices
0137 % E - F are the noise matrices
0138 % E - F are the constant input matrices
0139 options = plist('inputs', aos_varnames, 'outputs', outputs_varnames, 'states', 'ALL'
);
0140 [A B C D Ts aos_varnames2 ss_varnames2 outputs_varnames2] = double(copy(sys(i_sys),

```

```

true), options);
0141
0142 options = plist('inputs', noise_varnames, 'outputs', outputs_varnames, 'states', 'ALL'
);
0143 [A E C F Ts noise_varnames2 ss_varnames2 outputs_varnames2] = double(copy(sys(i_sys),
true), options);
0144
0145 options = plist('inputs', constants_varnames, 'outputs', outputs_varnames, 'states',
'ALL'
);
0146 [A G C H Ts constants_varnames2 ss_varnames2 outputs_varnames2] =
double(copy(sys(i_sys), true), options);
0147
0148 %% evaluating Kalman feedback K, innovation gain M, state covariance P, output
covariance Z
0149 % given Q and R (process and measurement noise covariances)
0150 Qn = E*noise_in*transpose(E);
0151 Qn = (Qn + 1e-10*norm(Qn)*eye(size(Qn)));
0152 Rn = F*noise_in*transpose(F);
0153 Rn = Rn + 1e-10*norm(Rn)*eye(size(Rn));
0154 Nn = E*noise_in*transpose(F);
0155 P = eye(size(A))*1e20;
0156 for i=1:1000
0157     P = A*P*A'+Qn;
0158     K = P*C'*(C*P*C'+Rn)^-1;
0159     P = (eye(size(A)) - K*C)*P;
0160 end
0161 Z = C*P*C' + Rn;
0162
0163 %% dividing or modifying size of matrices
0164 As = ssm.cell_recut(A, sssizes, sssizes);
0165 Bs = ssm.cell_recut(B, sssizes, Naos_in);
0166 Gs = ssm.cell_recut(G, sssizes, Nconstants_in);
0167 Cs = ssm.cell_recut(C, Noutputs_in, sssizes);
0168 Ks = ssm.cell_recut(K, sssizes, Noutputs_in);
0169
0170 %% getting data position in new matrices
0171
0172 for i=1:Naos_in
0173     % getting position
0174     [positions, sum, value] = ssm.cellstrfind(aos_varnames,aos_varnames2{i}, 'all');
0175     % checking against redundant input data
0176     if sum==1
0177         aos_pos(positions) = i;
0178         aos_data{positions} = aos_in(positions).data.getY ;
0179     elseif sum>1
0180         error(['more than one so variable is named ', value]);
0181     end
0182 end
0183 for i=1:Nconstants_in
0184     % getting position
0185     [positions, sum, value] =
ssm.cellstrfind(constants_varnames,constants_varnames2{i}, 'all');
0186     % checking against redundant input data
0187     if sum==1
0188         constants_pos(positions) = i;
0189     elseif sum>1
0190         error(['more than one constant variable is named ', value]);
0191     end
0192 end
0193 for i=1:Noutputs_in
0194     % getting position
0195     [positions, sum, value] =
ssm.cellstrfind(outputs_varnames,outputs_varnames2{i}, 'all');
0196     % checking against redundant output data
0197     if sum==1
0198         outputs_pos(positions) = i;
0199     elseif sum>1
0200         error(['more than one output variable is named ', value]);
0201     end
0202 end
0203
0204 %% getting correct number of samples
0205 Nsamples = inf;
0206 f0 = 1/timestep;
0207 for i=1:Naos_in
0208     Nsamples = min(Nsamples,length(aos_in(i).data.y));
0209     if ~(f0==aos_in(i).data.fs)
0210         str = ['WARNING : ssm frequency is ', num2str(f0), ...
' but sampling frequency of ao named '...
0211             aos_in(i).name, ' is ', num2str(aos_in(i).data.fs) ];
0212         utils.helper.msg(utils.const.msg.MNAME, str);
0213     end
0214 end
0215 % maybe tdata should be retrieved and verified to be equal, rather than this.
0216 end
0217 if Nsamples == inf % case there is no input!
0218     Nsamples = 0;

```

```

0219 end
0220
0221 %% evaluating data to return
0222 return_states = find(plist_inputs, 'return states');
0223 returnStates = false(1,Nss);
0224 for i=1:Nss
0225     returnStates(i) = ~isempty(find( ismember(return_states, i)));
0226 end
0227 return_outputs = find(plist_inputs, 'return outputs');
0228 returnOutputs = ~isempty(find( ismember(return_outputs, 1)));
0229
0230 %% simulation first step
0231 x = cell(Nss, 1);
0232 lastX = ssini;
0233 for i=1:Nss
0234     if returnStates(i)
0235         x{i} = zeros(sssizes(i), Nsamples);
0236     end
0237 end
0238 if returnOutputs
0239     y = zeros(Noutputs, Nsamples);
0240 end
0241 IncrementX = cell(Nss,1);
0242
0243 %% simulation loop
0244 for k = 1:Nsamples
0245     if rem(k,500)==0
0246         utils.helper.msg(utils.const.msg.MNAME, ['filtering time : ',num2str(k*timestep)
0247     ]);
0248 end
0249
0250 %% resetting fields
0251 for s = 1:Nss
0252     IncrementX{s} = zeros(sssizes(s),1);
0253 end
0254 IncrementY = zeros(Noutputs_in,1);
0255
0256 %% prediction step
0257 for i=1:Naos_in
0258     for s=1:Nss
0259         IncrementX{s} = IncrementX{s} + Bs{s}(:,aos_pos(i))*aos_data{i}(k);
0260     end
0261     IncrementY = IncrementY + D(:,aos_pos(i))*aos_data{i}(k);
0262 end
0263 for i=1:Nconstants_in
0264     for s=1:Nss
0265         IncrementX{s} = IncrementX{s} + Gs{s}(:,constants_pos(i))*constants_in(i);
0266     end
0267     IncrementY = IncrementY + H(:,constants_pos(i))*constants_in(i);
0268 end
0269
0270 % differential equation
0271 lastX = ssm.cell_add(IncrementX, ssm.cell_mult(As, lastX));
0272
0273 % output equations
0274 lastY = zeros(Noutputs_in,1);
0275 for s=1:Nss
0276     lastY = lastY + Cs{s}*lastX{s};
0277 end
0278 lastY = lastY + IncrementY;
0279
0280 thisY = zeros(Noutputs_in,1);
0281 for i=1:Noutputs_in
0282     thisY(outputs_pos(i)) = outputs_in(i).data.y(k);
0283 end
0284
0285 %% state correction (=update) step
0286 for i=1:Noutputs_in
0287     for s=1:Nss
0288         lastX{s} = lastX{s} + Ks{s}*( thisY - lastY) ;
0289     end
0290 end
0291
0292 % output correction
0293 lastY = zeros(Noutputs_in,1);
0294 for s=1:Nss
0295     lastY = lastY + Cs{s}*lastX{s};
0296 end
0297 lastY = lastY + IncrementY;
0298
0299 %% computing and storing outputs
0300 for s=1:Nss
0301     if returnStates(i)
0302         x{s}(:,k) = lastX{s};
0303     end
0304 end

```



```

0304     if returnOutputs
0305         y(:,k) = lastY;
0306     end
0307 end
0308 %% saving in aos
0309 Naos = 0;
0310 for i=1:Nss
0311     if returnStates(i)
0312         Naos = Naos + sssizes(i);
0313     end
0314 end
0315 if returnOutputs
0316     Naos = Naos + Noutputs_in;
0317 end
0318 ao_out(Naos) = ao;
0319 if ~exist('t','var')
0320     t = (1:Nsamples)*timestep;
0321 end
0322 count = 1;
0323 for i=1:Nss
0324     for j=1:sssizes(i)
0325         if returnStates(i)
0326             ao_out(count) = ao(tdata(t,x{i}(j,:)));
0327             ao_out(count).setName(['kal_X_',num2str(i),'_',num2str(j)]);
0328             ao_out(count).setDescription(...
0329                 ['ssm#:',num2str(i_sys),...
0330                  ', state:',sys(i_sys).ssnames{i},' ',sys(i_sys).ssvarnames{i}{j}]);
0331             ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs ,
0332                 {''}, sys(i_sys).hist );
0333             count = count+1;
0334         end
0335     end
0336 end
0337 for i=1:Noutputs_in
0338     if returnOutputs
0339         ao_out(count) = ao(tdata(t,y(i,:)));
0340         ao_out(count).setName(['kal_Y_',num2str(i)]);
0341         ao_out(count).setDescription(...
0342             ['ssm#:',num2str(i_sys),' ', output:', outputs_varnames2{i} ] );
0343         ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs ,
0344             {''}, sys(i_sys).hist );
0345         count = count+1;
0346     end
0347 end
0348 %% construct output analysis object
0349 plist_data_out = plist('K', K, 'P', P, 'Z', Z);
0350 varargout = {ao_out, plist_data_out};
0351 end
0352 function ii = getInfo(varargin)
0353 if nargin == 1 && strcmpi(varargin{1}, 'None')
0354     sets = {};
0355     pls = [];
0356 elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0357     sets{1} = varargin{1};
0358     pls = getDefaultPlist(sets{1});
0359 else
0360     sets = {'Default'};
0361     pls = [];
0362     for kk=1:numel(sets)
0363         pls = [pls getDefaultPlist(sets{kk})];
0364     end
0365 end
0366 % Build info object
0367 ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38
ingo Exp $', sets, pls);
0368 end
0369 function plo = getDefaultPlist(set)
0370 switch set
0371 case 'Default'
0372     plo = plist(...
0373         'noise variable names', {'noise_readout', 'noise_special'}, ...
0374         'variance', [1 0.1;1 0.1], ...
0375         'aos variable names', {'input_1','input_2'}, ...
0376         'aos', [ao, ao], ...
0377         'constant variable names', {'noise_readout' 'noise_readout2'}, ...
0378         'constant', [1;6], ...
0379         'output variable names', {'output_1'}, ...
0380         'outputs', ao , ...
0381         'return states', [1], ...
0382         'return outputs', [1], ...
0383         'ssini', {} ...
0384     );
0385 otherwise
0386     error('### Unknown parameter set [%s].', set);

```

0387 end

0388 end

minreal

PURPOSE ^

minreal gives a minimal realization of a ssm object by deleting state

SYNOPSIS ^

function varargout = minreal(varargin)

DESCRIPTION ^

```
minreal gives a minimal realization of a ssm object by deleting state
variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: minreal gives a minimal realization of a ssm object by
              deleting state variables

CALL:        Convert a statespace model object to a MATLAB statespace object.
              >> sys = minreal(sys);

              sys - an array of ssm object

VERSION:      $Id: ssm2ss.m,v 1.6 2008/06/08 18:44:07 adrien Exp $

M-FILE INFO:  Get information about this methods by calling
              >> ssm.getInfo('minreal')

              Get information about a specified set-plist by calling:
              >> ssm.getInfo('minreal', 'Default')

HISTORY: 20-08-2008 A Grynagier
          Creation
          TODO : Whole code will be rewritten. It turns out that the numerical
          discrepancies in the matlab minreal function case the whole system to
          become unstable!!!! I have to delete states 'by hand' or leave them as
          they are for now...
          I will code a function that will delete unreachable states but leave
          those that are reachable but pz cancelled.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [minreal](#) minreal gives a minimal realization of a ssm object by deleting state
-  [ssm2ss](#) SSM2SS converts a statespace model object to a MATLAB statespace object.
-  [validate](#)

This function is called by:

-  [minreal](#) minreal gives a minimal realization of a ssm object by deleting state
-  [ssm](#) SSM statespace model class constructor.
-  [ssm2iirpz](#) ssm2iirpz converts a statespace model object to an miir or a pzmodel

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 % minreal gives a minimal realization of a ssm object by deleting state
0002 % variables
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: minreal gives a minimal realization of a ssm object by
0006 %               deleting state variables
0007 %
0008 % CALL:        Convert a statespace model object to a MATLAB statespace object.
0009 %               >> sys = minreal(sys);
0010 %
0011 %               sys - an array of ssm object
0012 %
0013 %
0014 % VERSION:     $Id: ssm2ss.m,v 1.6 2008/06/08 18:44:07 adrien Exp $
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 %               >> ssm.getInfo('minreal')
0018 %
0019 %               Get information about a specified set-plist by calling:
0020 %               >> ssm.getInfo('minreal', 'Default')
0021 %
0022 % HISTORY: 20-08-2008 A Grynagier
0023 %           Creation
0024 % TODO : Whole code will be rewritten. It turns out that the numerical
0025 % discrepancies in the matlab minreal function case the whole system to
0026 % become unstable!!!! I have to delete states 'by hand' or leave them as
0027 % they are for now...
0028 % I will code a function that will delete unreachable states but leave
0029 % those that are reachable but pz cancelled.
0030 %
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 function varargout = minreal(varargin)
0034 %
0035 %% starting initial checks
0036 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0037 %
0038 % Check if this is a call for parameters
0039 if utils.helper.isinfocall(varargin{:})
0040     varargout{1} = getInfo(varargin{3});
0041     return
0042 end
0043 %
0044 % checking number of inputs work data
```

```

0045 if ~nargin==2, error('wrong number of inputs!'), end
0046
0047 if ~isequal(class(varargin{1}), 'ssm'), error(['argument is not a ssm but a ', class(varargin{1})]), end
0048
0049 sys = varargin{1};
0050
0051 %% begin function body
0052 for i_sys=1:length(sys)
0053     if ~sys(i_sys).isnumerical
0054         error('system should be numeric')
0055     end
0056     ss = ssm2ss(sys(i_sys), plist);
0057     ss = minreal(ss);
0058     [A,B,C] = ssdata(ss);
0059     sys(i_sys).amats = {A};
0060     sys(i_sys).mmats = {eye(size(A))};
0061
0062     bmats = cell(1, sys(i_sys).Ninputs);
0063     inputranks = cumsum([1, sys(i_sys).inputsizes]);
0064     for i=sys(i_sys).Ninputs
0065         bmats{i} = B(:, inputranks(i):(inputranks(i+1)-1));
0066     end
0067
0068     cmats = cell(sys(i_sys).Noutputs,1);
0069     outputranks = cumsum([1, sys(i_sys).outputsizes]);
0070     for i=sys(i_sys).Noutputs
0071         cmats{i} = C(outputranks(i):(outputranks(i+1)-1), :);
0072     end
0073
0074     sys(i_sys).bmats = bmats;
0075     sys(i_sys).cmats = cmats;
0076     sys(i_sys).amats_handles = cell(1,1);
0077     sys(i_sys).bmats_handles = cell(1, sys(i_sys).Ninputs);
0078     sys(i_sys).cmats_handles = cell(sys(i_sys).Noutputs,1);
0079     sys(i_sys).dmats_handles = {};
0080     sys(i_sys).sssizes = [];
0081     sys(i_sys).ssnames = sys(i_sys).ssnames(1);
0082     sys(i_sys).ssvarnames = {};
0083
0084     if (sys(i_sys).Nss > 1)
0085         utils.helper.msg(utils.const.msg.MNAME, ['warning in ', mfilename, ' : state space blocks will be fusionned!!!']);
0086     end
0087     sys(i_sys) = validate(sys(i_sys));
0088 end
0089 varargout{1} = sys;
0090
0091 end
0092
0093
0094 function ii = getInfo(varargin)
0095     if nargin == 1 && strcmpi(varargin{1}, 'None')
0096         sets = {};
0097         pls = [];
0098     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0099         sets{1} = varargin{1};
0100         pls = getDefaultPlist(sets{1});
0101     else
0102         sets = {'Default'};
0103         pls = [];
0104         for kk=1:numel(sets)
0105             pls = [pls getDefaultPlist(sets{kk})];
0106         end
0107     end
0108     % Build info object
0109     ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
0110 end
0111
0112 function pls = getDefaultPlist(set)
0113     switch set

```

Description of minreal

```
0114     case 'Default'
0115         plo = plist();
0116     otherwise
0117         error('### Unknown parameter set [%s].', set);
0118     end
0119 end
0120
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

modifparams

PURPOSE ^

modifparams enables to options(i_options)y and substitute parameters

SYNOPSIS ^

function varargout = modifparams(varargin)

DESCRIPTION ^

```
modifparams enables to options(i_options)y and substitute parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: modifparams enables to options(i_options)y and substitute parameters

CALL: varargout = modifparams(varargin)
[sys_out] = modifparams(sys, options)

INPUTS:
sys - (array of) ssm objects
options - plist of options:
'KEEP_PARAM' cell array of parameter to keep in the symbolic expression
'KEEP_PARAM_ORDER' array of 1 / 0 depending whether parameter are kept
or not
'SET_PARAM_NAMES' Cell array of parameter names for numerical
substitutions with user defined values
'SET_PARAM_VALUES' Array of parameter means for numerical substitutions
'SET_PARAM_SIGMAS' Array of parameter standard deviation for numerical
substitutions
'SET_PARAM_VALUES_ORDER' Array of 1 / 0 depending whether parameter
means are effectively substituted or not
'SET_PARAM_SIGMAS_ORDER' Array of 1 / 0 depending whether parameter
means are effectively substituted or not
'SYMBOLIC_OUTPUT_ALLOWED' if 0 checks output tensors are double and
not symbolic arrays

OUTPUTS:
The output array are of size Nsys*Noptions
sys_out - (array of) ssm objects with substitutions / modifications done
(case when more output than inputs)
sys - (array of) ssm objects with substitutions / modifications done
(case when the number of inputs and outputs is the same (numel
options =1))

M-FILE INFO: Get information about this methods by calling
>> ssm.getInfo('modifparams')

Get information about a specified set-plist by calling:
>> ssm.getInfo('modifparams', 'Default')

HISTORY: 17-04-2008 A Grynagier
01-04-2008 A Grynagier
26-02-2008 A Grynagier
29-01-2008 A Grynagier
```

```
Creation

TODO : Modify for multiple outputs
detect case when parameter is Set twice in a list
implement error message if keep is not empty and no symbolic output is
allowed
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY Make copy of ssm objects depending of the second input
-  **validate**
%%%

This function is called by:

-  **ssm** SSM statespace model class constructor.

SUBFUNCTIONS ^

-  **function ii** = getInfo(varargin)
-  **function plo** = getDefaultPlist(set)

SOURCE CODE ^

```
0001 % modifparams enables to options(i_options)y and substitute parameters
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: modifparams enables to options(i_options)y and substitute parameters
0005 %
0006 % CALL: varargout = modfiparams(varargin)
0007 % [sys_out] = modfiparams(sys, options)
0008 %
0009 % INPUTS:
0010 % sys - (array of) ssm objects
0011 % options - plist of options:
0012 % 'KEEP_PARAM' cell array of parameter to keep in the symbolic expression
0013 % 'KEEP_PARAM_ORDER' array of 1 / 0 depending whether parameter are kept
0014 % or not
0015 % 'SET_PARAM_NAMES' Cell array of parameter names for numerical
0016 % substitutions with user defined values
0017 % 'SET_PARAM_VALUES' Array of parameter means for numerical substitutions
0018 % 'SET_PARAM_SIGMAS' Array of parameter standard deviation for numerical
0019 % substitutions
0020 % 'SET_PARAM_VALUES_ORDER' Array of 1 / 0 depending whether parameter
0021 % means are effectively substituted or not
0022 % 'SET_PARAM_SIGMAS_ORDER' Array of 1 / 0 depending whether parameter
0023 % means are effectively substituted or not
0024 % 'SYMBOLIC_OUTPUT_ALLOWED' if 0 checks output tensors are double and
0025 % not symbolic arrays
0026 %
0027 % OUTPUTS:
0028 % The output array are of size Nsys*Noptions
0029 % sys_out - (array of) ssm objects with substitutions / modifications done
```

Description of modifparams

```

0030 % (case when more output than inputs)
0031 % sys - (array of) ssm objects with substitutions / modifications done
0032 % (case when the number of inputs and outputs is the same (numel
0033 % options =1))
0034 %
0035 % M-FILE INFO: Get information about this methods by calling
0036 % >> ssm.getInfo('modifparams')
0037 %
0038 % Get information about a specified set-plist by calling:
0039 % >> ssm.getInfo('modifparams', 'Default')
0040 %
0041 % HISTORY: 17-04-2008 A Grynagier
0042 % 01-04-2008 A Grynagier
0043 % 26-02-2008 A Grynagier
0044 % 29-01-2008 A Grynagier
0045 % Creation
0046 %
0047 % TODO : Modify for multiple outputs
0048 % detect case when parameter is Set twice in a list
0049 % implement error message if keep is not empty and no symbolic output is
0050 % allowed
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052
0053 function varargout = modifparams(varargin)
0054
0055 %% starting initial checks
0056 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0057
0058 % Check if this is a call for parameters
0059 if utils.helper.isinfocall(varargin{:})
0060     varargout{1} = getInfo(varargin{3});
0061     return
0062 end
0063
0064 % Collect input variable names
0065 in_names = cell(size(varargin));
0066 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0067
0068 % Collect all SSMs and plists
0069 [sys, ssm_invars] = utils.helper.collect_objects(varargin(:), 'ssm', in_names);
0070 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0071
0072 % Get defaults:
0073 options = combine(pl, getDefaultPlist('Default'));
0074
0075 Nsys = numel(sys);
0076 Noptions = numel(options);
0077 if Nsys ~= 1
0078     error('### Please input (only) one SSM model');
0079 end
0080
0081 % Decide on a deep copy or a modify, depending on the output
0082 sys = copy(sys, nargin);
0083
0084 %% begin function body
0085
0086 for i_sys = 1:Nsys
0087     for i_options = 1:Noptions
0088         options(i_options) = combine(options(i_options), getDefaultPlist('Default'));
0089         %% Retrieving option fields for operations
0090         KeepParamNamesIni = find(options(i_options), 'KEEP_PARAM_NAMES') ;
0091         KeepParamOrder = find(options(i_options), 'KEEP_PARAM_ORDER') ;
0092         SetParamNamesIni = find(options(i_options), 'SET_PARAM_NAMES') ;
0093         SetParamValuesIni = find(options(i_options), 'SET_PARAM_VALUES') ;
0094         SetParamSigmasIni = find(options(i_options), 'SET_PARAM_SIGMAS') ;
0095         SetParamValuesOrder = find(options(i_options), 'SET_PARAM_VALUES_ORDER') ;
0096         SetParamSigmasOrder = find(options(i_options), 'SET_PARAM_SIGMAS_ORDER') ;
0097         SymbolicOutputAllowed = find(options(i_options), 'SYMBOLIC_OUTPUT_ALLOWED') ;
0098         %% Retrieving system data

```

```

0099     if length(i_options)>1
0100         sys(i_sys, i_options) = sys(i_sys);
0101     end
0102
0103     %% building updated list for KeepParam...
0104     if isequal(KeepParamOrder, 'NONE') || SymbolicOutputAllowed==0
0105         KeepParamNames = {};
0106     elseif isequal(KeepParamOrder, 'LIST')
0107         KeepParamNames = KeepParamNamesIni;
0108     elseif isequal(KeepParamOrder, 'ALL')
0109         KeepParamNames = sys(i_sys).paramnames;
0110     else
0111         KeepParamNames = {};
0112         j=1;
0113         for i=1:length(KeepParamNamesIni)
0114             if KeepParamOrder(i)
0115                 KeepParamNames{j} = KeepParamNamesIni{i}; %#ok<AGROW>
0116                 j = j+1;
0117             end
0118         end
0119     end
0120     nKeep = length(KeepParamNames);
0121
0122     %% building updated list for SetParamValues
0123     if (isequal(SetParamValuesOrder, 'LIST'))
0124         SetParamValuesNames = SetParamNamesIni;
0125         SetParamValuesOrder = ones(length(SetParamNamesIni),1);
0126         SetParamValues = SetParamValuesIni;
0127     elseif (isequal(SetParamValuesOrder, 'NONE'))
0128         SetParamValuesNames = {};
0129         SetParamValuesOrder = [];
0130         SetParamValues = [];
0131     else
0132         SetParamValuesNames = SetParamNamesIni;
0133         SetParamValues = SetParamValuesIni;
0134     end
0135     nValues = length(SetParamValues);
0136
0137     %% building updated list for SetParamSigmas
0138     if (isequal(SetParamSigmasOrder, 'LIST'))
0139         SetParamSigmasNames = SetParamNamesIni;
0140         SetParamSigmasOrder = ones(length(SetParamNamesIni),1);
0141         SetParamSigmas = SetParamSigmasIni;
0142     elseif (isequal(SetParamSigmasOrder, 'NONE'))
0143         SetParamSigmasNames = {};
0144         SetParamSigmasOrder = [];
0145         SetParamSigmas = [];
0146     else
0147         SetParamSigmasNames = SetParamNamesIni;
0148         SetParamSigmas = SetParamSigmasIni;
0149     end
0150     nSigmas = length(SetParamSigmas);
0151
0152     %% building param position tables
0153     posValues = zeros(nValues,1);
0154     posKeep = zeros(nKeep,1);
0155     posSigmas = zeros(nSigmas,1);
0156     for i=1:sys(i_sys).Nparams
0157         for j=1:nKeep
0158             if isequal(sys(i_sys).paramnames{i},KeepParamNames{j})
0159                 posKeep(j) = i;
0160             end
0161         end
0162         for j=1:nValues
0163             if isequal(sys(i_sys).paramnames{i},SetParamValuesNames{j})
0164                 posValues(j) = i;
0165             end
0166         end
0167         for j=1:nSigmas

```



```

0168         if isequal(sys(i_sys).paramnames{i},SetParamSigmasNames{j})
0169             posSigmas(j) = i;
0170         end
0171     end
0172 end
0173
0174 %% looking for non existing but modified parameter names
0175 if not(nKeep==sum(posKeep>0))
0176     msg = 'error with keep parameter list because one does not exist in system.';
0177     error(msg);
0178 elseif not(nValues==sum(posValues>0))
0179     msg = 'error with SetValues parameter list because one does not exist in system.';
0180     error(msg);
0181 elseif not(nSigmas==sum(posSigmas>0))
0182     msg = 'error with SetSigma list because one does not exist in system.';
0183     error(msg);
0184 end
0185
0186 %% modifying parameter tables in sys(i_sys) (Setparam list)
0187 for i=1:nValues
0188     if SetParamValuesOrder(i)
0189         sys(i_sys).paramvalues(posValues(i)) = SetParamValues(i);
0190     end
0191 end
0192 for i=1:nSigmas
0193     if SetParamSigmasOrder(i)
0194         sys(i_sys).paramsigmas(posSigmas(i)) = SetParamSigmas(i);
0195     end
0196 end
0197
0198 %% Numerical value for symbolic variables
0199 for i=1:sys(i_sys).Nparams
0200     if sum(ismember(posKeep, i))==0
0201         cmd1 = ['sym(''', sys(i_sys).paramnames{i}, ''');'];
0202         cmd2 = [sys(i_sys).paramnames{i}, '= sys(i_sys).paramvalues(', num2str(i), ');'];
0203         eval(cmd1);
0204         eval(cmd2);
0205     else
0206         cmd1 = [sys(i_sys).paramnames{i}, '=sym(''', sys(i_sys).paramnames{i}, ''');'];
0207         eval(cmd1);
0208     end
0209 end
0210
0211 %% keeping only params in 'keep param names'
0212 sys(i_sys,i_options).paramnames = sys(i_sys,i_options).paramnames(posKeep);
0213 sys(i_sys,i_options).paramvalues = sys(i_sys,i_options).paramvalues(posKeep);
0214 sys(i_sys,i_options).paramsigmas = sys(i_sys,i_options).paramsigmas(posKeep);
0215
0216 %% evaluation of A, B, C, D matrices
0217 for i_ss=1:sys(i_sys,i_options).Nss
0218     for j_ss =1:sys(i_sys,i_options).Nss
0219         %Mass Matrix
0220         if ~isempty(sys(i_sys,i_options).mmats{i_ss, j_ss})
0221             if ~isnumeric(sys(i_sys,i_options).mmats{i_ss, j_ss})
0222                 sys(i_sys,i_options).mmats{i_ss, j_ss} = ...
0223                     eval(sym(sys(i_sys,i_options).mmats{i_ss, j_ss}));
0224             end
0225         end
0226         %A Matrix
0227         if ~isempty(sys(i_sys,i_options).amats{i_ss, j_ss})
0228             if ~isnumeric(sys(i_sys,i_options).amats{i_ss, j_ss})
0229                 sys(i_sys,i_options).amats{i_ss, j_ss} = ...
0230                     eval(sym(sys(i_sys,i_options).amats{i_ss, j_ss}));
0231             end
0232         end
0233     end
0234 end
0235 for i_ss=1:sys(i_sys,i_options).Nss
0236     for j_in =1:sys(i_sys,i_options).Ninputs

```

```

0237         %B Matrix
0238         if ~isempty(sys(i_sys,i_options).bmats{i_ss, j_in})
0239             if ~isnumeric(sys(i_sys,i_options).bmats{i_ss, j_in})
0240                 sys(i_sys,i_options).bmats{i_ss, j_in} = ...
0241                 eval(sym(sys(i_sys,i_options).bmats{i_ss, j_in}));
0242             end
0243         end
0244     end
0245 end
0246 for i_out=1:sys(i_sys,i_options).Noutputs
0247     for j_ss =1:sys(i_sys,i_options).Nss
0248         %C Matrix
0249         if ~isempty(sys(i_sys,i_options).cmats{i_out, j_ss})
0250             if ~isnumeric(sys(i_sys,i_options).cmats{i_out, j_ss})
0251                 sys(i_sys,i_options).cmats{i_out, j_ss} = ...
0252                 eval(sym(sys(i_sys,i_options).cmats{i_out, j_ss}));
0253             end
0254         end
0255     end
0256 end
0257 for i_out=1:sys(i_sys,i_options).Noutputs
0258     for j_in =1:sys(i_sys,i_options).Ninputs
0259         %D Matrix
0260         if ~isempty(sys(i_sys,i_options).dmats{i_out, j_in})
0261             if ~isnumeric(sys(i_sys,i_options).dmats{i_out, j_in})
0262                 sys(i_sys,i_options).dmats{i_out, j_in} = ...
0263                 eval(sym(sys(i_sys,i_options).dmats{i_out, j_in}));
0264             end
0265         end
0266     end
0267 end
0268 validate(sys(i_sys,i_options));
0269 sys(i_sys,i_options).addHistory(ssm.getInfo(mfilename), options(i_options) , {''}, sys(i_sys,i_options).hist );
0270 end
0271 end
0272 if nargout > 0
0273     varargout{1} = sys;
0274 end
0275 end
0276
0277
0278 %-----
0279 % Get Info Object
0280 %-----
0281 function ii = getInfo(varargin)
0282     if nargin == 1 && strcmpi(varargin{1}, 'None')
0283         sets = {};
0284         pls = [];
0285     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0286         sets{1} = varargin{1};
0287         pls = getDefaultPlist(sets{1});
0288     else
0289         sets = {'Default'};
0290         pls = [];
0291         for kk=1:numel(sets)
0292             pls = [pls getDefaultPlist(sets{kk})];
0293         end
0294     end
0295     % Build info object
0296     ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
0297 end
0298
0299 %-----
0300 % Get Default Plist
0301 %-----
0302 function plo = getDefaultPlist(set)
0303     switch set
0304     case 'Default'
0305         plo = plist( 'KEEP PARAM NAMES', {}, 'KEEP PARAM ORDER', 'LIST', ...

```

Description of modifparams

```
0306      'SET_PARAM_NAMES', {}, 'SET_PARAM_VALUES', [], 'SET_PARAM_SIGMAS', [], ...
0307      'SET_PARAM_VALUES_ORDER', 'LIST', 'SET_PARAM_SIGMAS_ORDER', 'NONE', 'SYMBOLIC_OUTPUT_ALLOWED', 1);
0308      otherwise
0309          error('### Unknown parameter set [%s].', set);
0310      end
0311 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

modifitimestep

PURPOSE ^

moditime modifies the timestep of a ssm object

SYNOPSIS ^

function varargout = modifitimestep(varargin)

DESCRIPTION ^

```
moditime modifies the timestep of a ssm object
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: moditime modifies the timestep of a ssm object, and updates
the A and B matrices suppsing there is no aliasing issue.

CALL: sys = makeSubsystem(sys,options)

INPUTS:
sys - (array of) ssm objects
options - (array of) plist giving new timestep value (param name 'newtimestep')

OUTPUTS: sys - (array of) ssm

***** THERE ARE NO DEFAULT PARAMETERS *****

M-FILE INFO: Get information about this methods by calling
              >> ssm.getInfo('modifitimestep')

              Get information about a specified set-plist by calling:
              >> ssm.getInfo('modifitimestep', 'Default')

VERSION: $Id: ltpda_ss_modify_time_setup.m,v 1.5 2008/03/11 16:52:56 adrien Exp $

HISTORY:
14-06-2008 M Weyrich changes for multiple options input
17-04-2008 A Grynagier
08-02-2008 A Grynagier
24-01-2008 A Grynagier
Creation 16-01-2008 A Grynagier

TODO : inplement history

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of ssm objects depending of the second input
-  [ssm](#) SSM statespace model class constructor.

This function is called by:

-  [ssm](#) SSM statespace model class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```

0001 % modifitime modifies the timestep of a ssm object
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: modifitime modifies the timestep of a ssm object, and updates
0005 % the A and B matrices suppsing there is no aliasing issue.
0006 %
0007 % CALL: sys = makeSubsystem(sys,options)
0008 %
0009 % INPUTS:
0010 % sys - (array of) ssm objects
0011 % options - (array of) plist giving new timestep value (param name 'newtimestep')
0012 %
0013 % OUTPUTS: sys - (array of) ssm
0014 %
0015 % ***** THERE ARE NO DEFAULT PARAMETERS *****
0016 %
0017 % M-FILE INFO: Get information about this methods by calling
0018 %                >> ssm.getInfo('modifitimestep')
0019 %
0020 %                Get information about a specified set-plist by calling:
0021 %                >> ssm.getInfo('modifitimestep', 'Default')
0022 %
0023 % VERSION: $Id: ltpda_ss_modify_time_setup.m,v 1.5 2008/03/11 16:52:56 adrien Exp $
0024 %
0025 % HISTORY:
0026 % 14-06-2008 M Weyrich changes for multiple options input
0027 % 17-04-2008 A Grynagier
0028 % 08-02-2008 A Grynagier
0029 % 24-01-2008 A Grynagier
0030 % Creation 16-01-2008 A Grynagier
0031 %
0032 % TODO : implement history
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035
0036 function varargout = modifitimestep(varargin)
0037
0038 %% starting initial checks
0039 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0040
0041 % Check if this is a call for parameters
0042 if utils.helper.isinfocall(varargin{:})
0043     varargout{1} = getInfo(varargin{3});
0044     return
0045 end
0046
0047 % checking number of inputs work data
0048 if ~nargin==2, error('wrong number of inputs!'), end
0049
0050 if ~isequal(class(varargin{1}), 'ssm'), error(['argument is not a ssm but a ',
class(varargin{1})]),end
0051 if ~isequal(class(varargin{2}), 'plist'), error(['argument is not a plist but a ',
class(varargin{2})]),end
0052
0053 sys_in = varargin{1};
0054 options = combine(varargin{2}, getDefaultPlist('Default'));
0055 Nsys = length(sys_in);
0056 Noptions = length(options);
0057 %% begin function body
0058
0059 %% building output depending on options
0060 if length(options)==1
0061     sys_out(:,1) = reshape(sys_in, length(sys_in),1);
0062 else
0063     for j=1:Noptions
0064         sys_out(:,j)=copy(reshape(sys_in, length(sys_in),1),true);
0065     end
0066 end
0067
0068 if ~(Noptions==1)
0069     sys_out(Nsys,Noptions) = ssm();
0070 end
0071
0072 for i = 1:Nsys
0073     for j = 1:Noptions
0074         %% retrieving input data
0075         options(j) = combine(options(j), getDefaultPlist('Default'));
0076         newtimestep = find(options(j), 'newtimestep');
0077         timestep = sys_in(i).timestep;
0078         sssizes = sys_in(i).sssizes;
0079         inputsizes = sys_in(i).inputsizes;
0080         amat = ssm.cell_fusion(sys_in(i).amats, sssizes, sssizes);

```

```

0081     mmat      = ssm.cell_fusion(sys_in(i).mmats, sssizes, sssizes);
0082     bmat      = ssm.cell_fusion(sys_in(i).bmats, sssizes, inputsizes);
0083     %% getting rid of M matrix
0084     minv = mmat^(-1);
0085     % minv = double(vpa(mmat,30)^(-1));
0086     amat = minv*amat;
0087     bmat = minv*bmat;
0088     mmat_2 = eye(size(amat));
0089     %% take different actions depending on old and new timestep
0090     if timestep == newtimestep
0091         action = 'DoNothing';
0092     elseif newtimestep == 0
0093         action = 'MakeContinuous';
0094         str = 'warning because system is sent back to continuous time';
0095         utils.helper.msg(utils.const.msg.MNAME, str);
0096     elseif timestep == 0
0097         action = 'Discretize';
0098     elseif floor(newtimestep/timestep) == newtimestep/timestep
0099         action = 'TakeMultiple';
0100     elseif newtimestep > timestep
0101         action = 'MakeLonger';
0102     else
0103         action = 'MakeShorter';
0104         str = 'warning because system is sent back to shorter time step';
0105         utils.helper.msg(utils.const.msg.MNAME, str);
0106     end
0107
0108     %% proceed with matrix modifications
0109     NSubstep = 35;
0110     amat_input = zeros(size(amat));
0111     if isequal(action, 'DoNothing')
0112         %% same timestep : do nothing
0113         amat_2 = amat;
0114         bmat_2 = bmat;
0115     elseif isequal(action, 'Discretize')
0116         %% from continuous to discrete
0117         amat_2 = expm(amat * newtimestep);
0118         for i_step = 1:NSubstep
0119             position = newtimestep*(2*i_step-1)/(2*NSubstep);
0120             amat_input = amat_input + expm(amat*position);
0121         end
0122         amat_input = amat_input*newtimestep/NSubstep;
0123         bmat_2 = real(amat_input * bmat);
0124     elseif isequal(action, 'MakeContinuous')
0125         %% for discrete to continuous
0126         [V, E] = eig(amat);
0127         amat_2 = V * (diag(diag(log(E)))/timestep) * V^(-1);
0128         for i_step = 1:NSubstep
0129             position = timestep*(2*i_step-1)/(2*NSubstep);
0130             amat_input = amat_input + expm(amat_2*position);
0131         end
0132         amat_input = amat_input*timestep/NSubstep;
0133         amat_input_inv = (amat_input)^(-1);
0134         bmat_2 = real(amat_input_inv * bmat);
0135     elseif isequal(action, 'TakeMultiple')
0136         %% discrete to discrete with a multiple
0137         multiple = newtimestep/timestep;
0138         amat_2 = amat^multiple;
0139         for i_step = 1:multiple
0140             amat_input = amat_input + amat^(i_step-1);
0141         end
0142         bmat_2 = real(amat_input * bmat);
0143
0144     elseif isequal(action, 'MakeLonger') || isequal(action, 'MakeShorter')
0145         %% discrete to discrete with no multiple relationship
0146         [V, E] = eig(amat);
0147         amat_1 = V * (diag(diag(log(E)))/timestep) * V^(-1);
0148         amat_2 = expm(amat_1 * newtimestep);
0149         amat_input_1 = amat_input;
0150         amat_input_2 = amat_input;
0151         for i_step = 1:NSubstep
0152             position_1 = timestep*(2*i_step-1)/(2*NSubstep);
0153             position_2 = newtimestep*(2*i_step-1)/(2*NSubstep);
0154             amat_input_1 = amat_input_1 + expm(amat_1*position_1);
0155             amat_input_2 = amat_input_2 + expm(amat_1*position_2);
0156         end
0157         amat_input_1 = amat_input_1*timestep/NSubstep;
0158         amat_input_2 = amat_input_2*newtimestep/NSubstep;
0159         amat_input_1inv = (amat_input_1)^(-1);
0160         bmat_2 = real(amat_input_2 * amat_input_1inv * bmat);
0161     end
0162     amat_2 = real(amat_2);
0163     bmat_2 = real(bmat_2);
0164     %% Save Matrix modifications
0165     sys_out(i,j).amats = ssm.cell_recut(amat_2, sssizes, sssizes);
0166     sys_out(i,j).mmats = ssm.cell_recut(mmat_2, sssizes, sssizes);

```

```

0167     sys_out(i,j).bmats = ssm.cell_recut(bmat_2, sssizes, inputsizes);
0168     sys_out(i,j).timestep = newtimestep;
0169     sys_out(i,j).addHistory(ssm.getInfo(mfilename), options(j) , {''}, sys_in(i).hist
);
0170 end
0171 end
0172 if ~(Noptions==1)
0173     varargout = {sys_out};
0174 else
0175     varargout = {sys_in};
0176 end
0177 end
0178
0179 function ii = getInfo(varargin)
0180     if nargin == 1 && strcmpi(varargin{1}, 'None')
0181         sets = {};
0182         pls = [];
0183     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0184         sets{1} = varargin{1};
0185         pls = getDefaultPlist(sets{1});
0186     else
0187         sets = {'Default'};
0188         pls = [];
0189         for kk=1:numel(sets)
0190             pls = [pls getDefaultPlist(sets{kk})];
0191         end
0192     end
0193 end
0194 % Build info object
0195 ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38
ingo Exp $', sets, pls);
0196 end
0197
0198 function plo = getDefaultPlist(set)
0199     switch set
0200     case 'Default'
0201         plo = plist('newtimestep',0.1);
0202     otherwise
0203         error('### Unknown parameter set [%s].', set);
0204     end
0205 end
0206

```

modify

PURPOSE ^

modify allows to exectue a string to modify a ssm object

SYNOPSIS ^

function varargout = modify(varargin)

DESCRIPTION ^

modify allows to exectue a string to modify a ssm object
%%%

DESCRIPTION: modify allows to exectue a tring to modify a ssm object

CALL: varargout = modify(varargin)

INPUTS:

sys - (array of) ssm objects

options - plist of options:

 'commands' cell array commands to execute

 typical command should be of the type :

 'amats{1,1}(2,3) = -6;'

 and will be executed as :

 'sys(i_sys).amats{1,1}(2,3) = -6;'

 they include no ssm reference, but the semicolon.

OUTPUTS:

 The output array are of size Nsys*Noptions

sys_out - (array of) ssm objects without the modifs

HISTORY: 05-08-2008 A Grynagier Creation

M-FILE INFO: Get information about this methods by calling

 >> ssm.getInfo('modify')

 Get information about a specified set-plist by calling:

 >> ssm.getInfo('modify', 'Default')

TODO :

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

 **copy** COPY Make copy of ssm objects depending of the second input

 **validate**

%%%

This function is called by:

 **ssm** SSM statespace model class constructor.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist(set)

SOURCE CODE ^

```
0001 % modify allows to exectue a string to modify a ssm object
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: modify allows to exectue a tring to modify a ssm object
0005 %
0006 % CALL: varargout = modify(varargin)
0007 %
0008 % INPUTS:
0009 % sys - (array of) ssm objects
0010 % options - plist of options:
0011 % 'commands' cell array commands to execute
0012 % typical command should be of the type :
0013 % 'amats{1,1}(2,3) = -6;'
0014 % and will be executed as :
0015 % 'sys(i_sys).amats{1,1}(2,3) = -6;'
0016 % they include no ssm reference, but the semicolon.
0017 %
0018 % OUTPUTS:
0019 % The output array are of size Nsys*Noptions
0020 % sys_out - (array of) ssm objects without the modifs
0021 %
0022 % HISTORY: 05-08-2008 A Grynagier Creation
0023 %
0024 % M-FILE INFO: Get information about this methods by calling
0025 % >> ssm.getInfo('modify')
0026 %
0027 % Get information about a specified set-plist by calling:
0028 % >> ssm.getInfo('modify', 'Default')
0029 %
0030 % TODO :
0031 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032 %
0033 function varargout = modify(varargin)
0034
0035 %% starting initial checks
0036 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0037
0038 % Check if this is a call for parameters
0039 if utils.helper.isinfocall(varargin{:})
0040     varargout{1} = getInfo(varargin{3});
0041     return
0042 end
0043
0044 % Collect input variable names
0045 in_names = cell(size(varargin));
0046 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0047
0048 % Collect all SSMS and plists
0049 [sys, ssm_invars] = utils.helper.collect_objects(varargin(:), 'ssm', in_names);
0050 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
```

```

0051
0052 options = combine(pl, getDefaultPlist('Default'));
0053 Nsys      = numel(sys);
0054 Noptions  = numel(options);
0055
0056 % Decide on a deep copy or a modify, depending on the output
0057 sys = copy(sys, nargout);
0058
0059 %% begin function body
0060
0061 for i_sys=1:Nsys
0062     for i_options=1:Noptions
0063         commands = find(options(i_options), 'commands');
0064         for i_commands=1:length(commands)
0065             strng = ['sys(i_sys,i_options).', commands{i_commands}];
0066             utils.helper.msg(utils.const.msg.MNAME, 'executing command: ');
0067             utils.helper.msg(utils.const.msg.MNAME, strng);
0068             eval(strng);
0069         end
0070         sys(i_sys,i_options) = validate(sys(i_sys,i_options));
0071         sys(i_sys,i_options).addHistory(ssm.getInfo(mfilename), options(i_options) , {''}, sys(i_sys).hist );
0072     end
0073 end
0074 if nargout > 0
0075     varargout{1} = sys;
0076 end
0077 end
0078
0079
0080 %-----
0081 % Get Info Object
0082 %-----
0083 function ii = getInfo(varargin)
0084     if nargin == 1 && strcmpi(varargin{1}, 'None')
0085         sets = {};
0086         pls = [];
0087     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0088         sets{1} = varargin{1};
0089         pls = getDefaultPlist(sets{1});
0090     else
0091         sets = {'Default'};
0092         pls = [];
0093         for kk=1:numel(sets)
0094             pls = [pls getDefaultPlist(sets{kk})];
0095         end
0096     end
0097     % Build info object
0098     ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
0099 end
0100
0101 %-----
0102 % Get Default Plist
0103 %-----
0104 function plo = getDefaultPlist(set)
0105     switch set
0106     case 'Default'
0107         plo = plist( 'commands', {'=ssm;'});
0108     otherwise
0109         error('### Unknown parameter set [%s].', set);
0110     end
0111 end
0112

```

reduce

PURPOSE ^

reduce enables to do model simplification

SYNOPSIS ^

function varargout = reduce(varargin)

DESCRIPTION ^

```
reduce enables to do model simplification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: reduce enables to do model simplification by erasing some state
spaces/inputs/outputs

CALL: varargout = reduce(varargin)

INPUTS:
sys - (array of) ssm objects
options - plist of options:
    'ERASE_INPUTS' cell array inputs to delete
    'ERASE_STATES' cell array states to delete
    'ERASE_OUTPUTS' cell array outputs to delete

OUTPUTS:
    The output array are of size Nsys*Noptions
sys_out - (array of) ssm objects without the specified information

M-FILE INFO: Get information about this methods by calling
    >> ssm.getInfo('reduce')

            Get information about a specified set-plist by calling:
    >> ssm.getInfo('reduce', 'Default')

HISTORY: 05-08-2008 A Grynagier Creation

TODO :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of ssm objects depending of the second input
-  [validate](#)

%%

This function is called by:

 **ssm** SSM statespace model class constructor.

SUBFUNCTIONS ^

-  `function ii = getInfo(varargin)`
-  `function plo = getDefaultPlist(set)`

SOURCE CODE ^

```
0001 % reduce enables to do model simplification
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: reduce enables to do model simplification by erasing some state
0005 % spaces/inputs/outputs
0006 %
0007 % CALL: varargout = reduce(varargin)
0008 %
0009 % INPUTS:
0010 % sys - (array of) ssm objects
0011 % options - plist of options:
0012 % 'ERASE_INPUTS' cell array inputs to delete
0013 % 'ERASE_STATES' cell array states to delete
0014 % 'ERASE_OUTPUTS' cell array outputs to delete
0015 %
0016 % OUTPUTS:
0017 % The output array are of size Nsys*Noptions
0018 % sys_out - (array of) ssm objects without the specified information
0019 %
0020 % M-FILE INFO: Get information about this methods by calling
0021 % >> ssm.getInfo('reduce')
0022 %
0023 % Get information about a specified set-plist by calling:
0024 % >> ssm.getInfo('reduce', 'Default')
0025 %
0026 % HISTORY: 05-08-2008 A Grynagier Creation
0027 %
0028 % TODO :
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 function varargout = reduce(varargin)
0032
0033 % starting initial checks
0034 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0035
0036 % Check if this is a call for parameters
0037 if utils.helper.isinfocall(varargin{:})
0038     varargout{1} = getInfo(varargin{3});
0039     return
0040 end
0041
0042 % checking number of inputs work data
0043 if ~nargin==2, error('wrong number of inputs!'), end
0044
0045 if ~isequal(class(varargin{1}), 'ssm'), error(['argument is not a ssm but a ', class(varargin{1})]), end
0046 if ~isequal(class(varargin{2}), 'plist'), error(['argument is not a plist but a ', class(varargin{2})]), end
0047
0048 sys = varargin{1};
0049 options = varargin{2};
0050 Nsys = length(sys);
0051 Noptions = length(options);
0052 % begin function body
```

```

0053
0054 % building output depending on options
0055 if Noptions==1
0056     sys_out(:,1) = reshape(sys,Nsys,1);
0057 else
0058     for j=1:Noptions
0059         sys_out(:,j)=copy(reshape(sys,Nsys,1),true);
0060     end
0061 end
0062 for i_sys=1:Nsys
0063     for i_options=1:Noptions
0064         erase_inputs = find(options(i_options), 'ERASE_INPUTS');
0065         erase_states = find(options(i_options), 'ERASE_STATES');
0066         erase_outputs = find(options(i_options), 'ERASE_OUTPUTS');
0067         for i=1:length(erase_inputs)
0068             [pos, sum, value] = ssm.cellstrfind(sys_out(i_sys, i_options).inputnames, erase_inputs{i}, 'all');
0069             if sum==0
0070                 error(['input to delete named ',erase_inputs{i}, ' was not found in system ', sys_out(i_sys, i_options).name])
0071             elseif sum>1
0072                 error(['input to delete named ',erase_inputs{i}, ' was found more than once in system ', sys_out(i_sys, i_options).name])
0073             else
0074                 Ninputs = sys_out(i_sys, i_options).Ninputs;
0075                 sys_out(i_sys, i_options).bmats = [ sys_out(i_sys, i_options).bmats(:, 1:pos-1) sys_out(i_sys, i_options).bmats(:,
pos+1:Ninputs) ];
0076                 sys_out(i_sys, i_options).dmats = [ sys_out(i_sys, i_options).dmats(:, 1:pos-1) sys_out(i_sys, i_options).dmats(:,
pos+1:Ninputs) ];
0077                 sys_out(i_sys, i_options).bmats_handles = [ sys_out(i_sys, i_options).bmats_handles(:, 1:pos-1) sys_out(i_sys,
i_options).bmats_handles(:, pos+1:Ninputs) ];
0078                 sys_out(i_sys, i_options).dmats_handles = [ sys_out(i_sys, i_options).dmats_handles(:, 1:pos-1) sys_out(i_sys,
i_options).dmats_handles(:, pos+1:Ninputs) ];
0079                 sys_out(i_sys, i_options).inputnames = [ sys_out(i_sys, i_options).inputnames(1:pos-1) sys_out(i_sys,
i_options).inputnames(pos+1:Ninputs) ];
0080                 sys_out(i_sys, i_options).inputvarnames = [ sys_out(i_sys, i_options).inputvarnames(1:pos-1) sys_out(i_sys,
i_options).inputvarnames(pos+1:Ninputs) ];
0081                 sys_out(i_sys, i_options).inputsizes = [];
0082             end
0083         end
0084         for i=1:length(erase_outputs)
0085             [pos, sum, value] = ssm.cellstrfind(sys_out(i_sys, i_options).outputnames, erase_outputs{i}, 'all');
0086             if sum==0
0087                 error(['input to delete named ',erase_outputs{i}, ' was not found in system ', sys_out(i_sys, i_options).name])
0088             elseif sum>1
0089                 error(['input to delete named ',erase_outputs{i}, ' was found more than once in system ', sys_out(i_sys, i_options).name])
0090             else
0091                 Noutputs = sys_out(i_sys, i_options).Noutputs;
0092                 sys_out(i_sys, i_options).cmats = [ sys_out(i_sys, i_options).cmats(1:pos-1, :) ; sys_out(i_sys,
i_options).cmats(pos+1:Noutputs, :) ];
0093                 sys_out(i_sys, i_options).dmats = [ sys_out(i_sys, i_options).dmats(1:pos-1, :) ; sys_out(i_sys,
i_options).dmats(pos+1:Noutputs, :) ];
0094                 sys_out(i_sys, i_options).cmats_handles = [ sys_out(i_sys, i_options).cmats_handles(1:pos-1, :) ; sys_out(i_sys,
i_options).cmats_handles(pos+1:Noutputs, :) ];
0095                 sys_out(i_sys, i_options).dmats_handles = [ sys_out(i_sys, i_options).dmats_handles(1:pos-1, :) ; sys_out(i_sys,
i_options).dmats_handles(pos+1:Noutputs, :) ];
0096                 sys_out(i_sys, i_options).outputnames = [ sys_out(i_sys, i_options).outputnames(1:pos-1) sys_out(i_sys,
i_options).outputnames(pos+1:Noutputs) ];
0097                 sys_out(i_sys, i_options).outputvarnames = [ sys_out(i_sys, i_options).outputvarnames(1:pos-1) sys_out(i_sys,
i_options).outputvarnames(pos+1:Noutputs) ];
0098                 sys_out(i_sys, i_options).outputsizes = [];
0099             end
0100         end
0101         for i=1:length(erase_states)
0102             [pos, sum, value] = ssm.cellstrfind(sys_out(i_sys, i_options).ssnames, erase_states{i}, 'all');
0103             if sum==0
0104                 error(['input to delete named ',erase_states{i}, ' was not found in system ', sys_out(i_sys, i_options).name])
0105             elseif sum>1
0106                 error(['input to delete named ',erase_states{i}, ' was found more than once in system ', sys_out(i_sys, i_options).name])
0107             else
0108                 Nss = sys_out(i_sys, i_options).Nss;
0109                 sys_out(i_sys, i_options).amats = [...

```

Description of reduce

```

0110         sys_out(i_sys, i_options).amats(1:pos-1, 1:pos-1)      sys_out(i_sys, i_options).amats(1:pos-1, pos+1:Nss) ;...
0111         sys_out(i_sys, i_options).amats(pos+1:Nss, 1:pos-1)    sys_out(i_sys, i_options).amats(pos+1:Nss, pos+1:Nss)];
0112         sys_out(i_sys, i_options).mmats = [...
0113         sys_out(i_sys, i_options).mmats(1:pos-1, 1:pos-1)      sys_out(i_sys, i_options).mmats(1:pos-1, pos+1:Nss) ;...
0114         sys_out(i_sys, i_options).mmats(pos+1:Nss, 1:pos-1)    sys_out(i_sys, i_options).mmats(pos+1:Nss, pos+1:Nss)];
0115         sys_out(i_sys, i_options).bmats = [ sys_out(i_sys, i_options).bmats(1:pos-1, :) ; sys_out(i_sys,
i_options).bmats(pos+1:Nss, :) ] ;
0116         sys_out(i_sys, i_options).cmats = [ sys_out(i_sys, i_options).cmats(:, 1:pos-1) sys_out(i_sys, i_options).cmats(:,
pos+1:Nss) ] ;
0117         sys_out(i_sys, i_options).amats_handles = [...
0118         sys_out(i_sys, i_options).amats_handles(1:pos-1, 1:pos-1)      sys_out(i_sys, i_options).amats_handles(1:pos-1,
pos+1:Nss) ;...
0119         sys_out(i_sys, i_options).amats_handles(pos+1:Nss, 1:pos-1)    sys_out(i_sys, i_options).amats_handles(pos+1:Nss,
pos+1:Nss)];
0120         sys_out(i_sys, i_options).bmats_handles = [ sys_out(i_sys, i_options).bmats_handles(1:pos-1, :) ; sys_out(i_sys,
i_options).bmats_handles(pos+1:Nss, :) ] ;
0121         sys_out(i_sys, i_options).cmats_handles = [ sys_out(i_sys, i_options).cmats_handles(:, 1:pos-1) sys_out(i_sys,
i_options).cmats_handles(:, pos+1:Nss) ] ;
0122         sys_out(i_sys, i_options).ssnames = [ sys_out(i_sys, i_options).ssnames(1:pos-1) sys_out(i_sys,
i_options).ssnames(pos+1:Nss) ] ;
0123         sys_out(i_sys, i_options).ssvarnames = [ sys_out(i_sys, i_options).ssvarnames(1:pos-1) sys_out(i_sys,
i_options).ssvarnames(pos+1:Nss)];
0124         sys_out(i_sys, i_options).sssizes = [];
0125     end
0126 end
0127 sys_out(i_sys, i_options) = validate(sys_out(i_sys, i_options));
0128 sys_out(i_sys, i_options).addHistory(ssm.getInfo(mfilename), options(i_options) , {''}, sys(i_sys).hist );
0129 end
0130 end
0131 varargout = {sys_out};
0132 end
0133
0134
0135 function ii = getInfo(varargin)
0136     if nargin == 1 && strcmpi(varargin{1}, 'None')
0137         sets = {};
0138         pls = [];
0139     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0140         sets{1} = varargin{1};
0141         pls = getDefaultPlist(sets{1});
0142     else
0143         sets = {'Default'};
0144         pls = [];
0145         for kk=1:numel(sets)
0146             pls = [pls getDefaultPlist(sets{kk})];
0147         end
0148     end
0149 % Build info object
0150 ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
0151 end
0152
0153 function plo = getDefaultPlist(set)
0154     switch set
0155     case 'Default'
0156         plo = plist('ERASE_STATES', {'state 1'}, ...
0157         'ERASE_INPUTS', {'INPUT 2'}, ...
0158         'ERASE_OUTPUTS', 'OUTPUT 3');
0159     otherwise
0160         error('### Unknown parameter set [%s].', set);
0161     end
0162 end
0163

```

reduce_model

PURPOSE ^

REDUCE_MODEL enables to do model simplification

SYNOPSIS ^

function varargout = reduce_model(varargin)

DESCRIPTION ^

```
REDUCE_MODEL enables to do model simplification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: reduce_model allows to eliminate state space variables.

CALL:      [ssm] = reduce_model(ssm, options);

INPUTS :
    ssm      - a ssm object
    options  - an options plist

OPTIONS :
plist with parameters 'inputs', 'states' and 'outputs' to indicate which
inputs, states and outputs variables are taken in account. This requires
proper variable naming. If a variable called appears more that once it
will be used once only.
The field may be :
    - a cellstr containing the resp. input/state/output *variable* names
    - a logical indexing the resp. input/state/output
      *variables*. Index is stored in a cell array, each cell
      correponding to one input/state/output block.
    - a double indexing the resp. input/state/output
      *variables*. Index is stored in a cell array, each cell
      correponding to one input/state/output block.
    - 'ALL', this string indicates all i/o variables will be
      given

OUTPUTS:
    The output array are of size Nsys*Noptions
sys_out - (array of) ssm objects without the specified information

M-FILE INFO: Get information about this methods by calling
    >> ssm.getInfo('reduce_model')

    Get information about a specified set-plist by calling:
    >> ssm.getInfo('reduce_model', 'Default')

HISTORY:  05-08-2008 A Grynagier Creation

TODO :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of ssm objects depending of the second input

This function is called by:

-  [double](#) Convert a statespace model object to double arrays for given i/o
-  [ssm](#) SSM statespace model class constructor.



SUBFUNCTIONS

 function ii = getInfo(varargin)
 function plo = getDefaultPlist(set)

SOURCE CODE ^

```

0001 % REDUCE_MODEL enables to do model simplification
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 %
0005 % DESCRIPTION: reduce_model allows to eliminate state space variables.
0006 %
0007 % CALL:          [ssm] = reduce_model(ssm, options);
0008 %
0009 % INPUTS :
0010 %          ssm      - a ssm object
0011 %          options  - an options plist
0012 %
0013 % OPTIONS :
0014 % plist with parameters 'inputs', 'states' and 'outputs' to indicate which
0015 % inputs, states and outputs variables are taken in account. This requires
0016 % proper variable naming. If a variable called appears more that once it
0017 % will be used once only.
0018 % The field may be :
0019 %          - a cellstr containing the resp. input/state/output *variable* names
0020 %          - a logical indexing the resp. input/state/output
0021 %            *variables*. Index is stored in a cell array, each cell
0022 %            corresponding to one input/state/output block.
0023 %          - a double indexing the resp. input/state/output
0024 %            *variables*. Index is stored in a cell array, each cell
0025 %            corresponding to one input/state/output block.
0026 %          - 'ALL', this string indicates all i/o variables will be
0027 %            given
0028 %
0029 % OUTPUTS:
0030 %          The output array are of size Nsys*Noptions
0031 %          sys_out - (array of) ssm objects without the specified information
0032 %
0033 % M-FILE INFO: Get information about this methods by calling
0034 %               >> ssm.getInfo('reduce_model')
0035 %
0036 %               Get information about a specified set-plist by calling:
0037 %               >> ssm.getInfo('reduce_model', 'Default')
0038 %
0039 % HISTORY:  05-08-2008 A Grynagier Creation
0040 %
0041 % TODO :
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043 %
0044 function varargout = reduce_model(varargin)
0045 %
0046 import utils.const.*
0047 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0048 %% starting initial checks
0049 %
0050 % Check if this is a call for parameters
0051 if utils.helper.isinfocall(varargin{:})
0052     varargout{1} = getInfo(varargin{3});
0053     return
0054 end
0055 %
0056 % checking number of inputs work data
0057 if ~nargin==2, error('wrong number of inputs!'), end
0058 %
0059 % Collect input variable names
0060 in_names = cell(size(varargin));
0061 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0062 %
0063 % Collect all SSMs and plists
0064 [sys, ssm_invars] = utils.helper.collect_objects(varargin(:), 'ssm', in_names);
0065 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0066 %
0067 % Get defaults:
0068 options = combine(pl, getDefaultPlist('Default'));
0069 %
0070 Nsys = numel(sys);
0071 Noptions = numel(options);
0072 if Nsys ~= 1
0073     error('### Please input (only) one SSM model');

```



```

0075 end
0076
0077 % Decide on a deep copy or a modify, depending on the output
0078 sys = copy(sys, nargout);
0079
0080 %% begin function body
0081
0082 % Loop over input systems
0083 for i_sys=1:Nsys
0084     for i_options=1:Noptions
0085         options(i_options) = combine(options(i_options), getDefaultPlist('Default'));
0086         inputs = find(options(i_options), 'inputs');
0087         outputs = find(options(i_options), 'outputs');
0088         states = find(options(i_options), 'states');
0089
0090         % Support single string inputs and single number inputs
0091         if ischar(inputs) || isnumeric(inputs), inputs = {inputs}; end
0092         if ischar(outputs) || isnumeric(outputs), outputs = {outputs}; end
0093         if ischar(states) || isnumeric(states), states = {states}; end
0094
0095         %% converting the lists of the i/o wanted - ss is not modified
0096         %
0097         % widths      lines wanted      cols wanted      line names      cell_mat      heights
0098         % [As]          states,          states,          = ssm.cell_select(sys.amats, sys.sssizes,
0099         % sys.sssizes,   states,          states,          sys.ssvarnames, sys.ssvarnames);
0100         % [Ms]          states,          states,          = ssm.cell_select(sys.mmats, sys.sssizes,
0101         % sys.sssizes,   states,          states,          sys.ssvarnames, sys.ssvarnames);
0102         % [Bs, sssizes] states,          inputs,         = ssm.cell_select(sys.bmats, sys.sssizes,
0103         % sys.inputsizes, states,          inputs,         sys.ssvarnames, sys.inputvarnames);
0104         % [Cs]          outputs,          states,         = ssm.cell_select(sys.cmats, sys.outputsizes,
0105         % sys.sssizes,   outputs,          states,         sys.outputvarnames, sys.ssvarnames);
0106         % [Ds, outputsizes, inputsizes] = ssm.cell_select(sys.dmats, sys.outputsizes,
0107         % sys.inputsizes, outputs,          inputs,         sys.outputvarnames, sys.inputvarnames);
0108         % [inputvarnames2] = ssm.cell_select(sys.inputvarnames, 1,
0109         % sys.inputsizes, {true},          inputs,         {''}, sys.inputvarnames);
0110         % [ssvarnames2]   = ssm.cell_select(sys.ssvarnames, 1,
0111         % sys.sssizes,    {true},          states,         {''}, sys.ssvarnames);
0112         % [outputvarnames2] = ssm.cell_select(sys.outputvarnames, 1,
0113         % sys.outputsizes, {true},         outputs,        {''}, sys.outputvarnames);
0114
0115         sys(i_sys, i_options).amats = As;
0116         sys(i_sys, i_options).mmats = Ms;
0117         sys(i_sys, i_options).bmats = Bs;
0118         sys(i_sys, i_options).cmats = Cs;
0119         sys(i_sys, i_options).dmats = Ds;
0120         sys(i_sys, i_options).inputvarnames = inputvarnames2;
0121         sys(i_sys, i_options).ssvarnames = ssvarnames2;
0122         sys(i_sys, i_options).outputvarnames = outputvarnames2;
0123         sys(i_sys, i_options).inputsizes = inputsizes;
0124         sys(i_sys, i_options).sssizes = sssizes;
0125         sys(i_sys, i_options).outputsizes = outputsizes;
0126
0127         sys(i_sys, i_options).name = [sys(i_sys, i_options).name, ' reduced'];
0128         sys(i_sys, i_options).description = [sys(i_sys, i_options).description, '
0129 reduced'];
0130         sys(i_sys, i_options).addHistory(ssm.getInfo(mfilename), options(i_options),
0131 {''}, sys(i_sys, i_options).hist);
0132     end
0133 end
0134 sys.validate;
0135 if nargout > 0
0136     varargout{1} = sys;
0137 end
0138 end
0139
0140 %-----
0141 % Get Info Object
0142 %-----
0143 function ii = getInfo(varargin)
0144 if nargin == 1 && strcmpi(varargin{1}, 'None')
0145     sets = {};
0146     pls = [];
0147 elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0148     sets{1} = varargin{1};
0149     pls = getDefualtPlist(sets{1});
0150 else
0151     sets = {'Default'};
0152     pls = [];
0153     for kk=1:numel(sets)
0154         pls = [pls getDefualtPlist(sets{kk})];
0155     end
0156 end
0157 % Build info object
0158 ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v
0159 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);

```

```
0149 end
0150
0151 %-----
0152 % Get Default Plist
0153 %-----
0154 function plo = getDefaultPlist(set)
0155     switch set
0156     case 'Default'
0157         plo = plist('inputs','ALL','outputs','ALL','states','ALL');
0158     otherwise
0159         error('### Unknown parameter set [%s].', set);
0160     end
0161 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

simulate

PURPOSE ^

simulate simulates a discrete ssm with given inputs

SYNOPSIS ^

function varargout = simulate(varargin)

DESCRIPTION ^

```
simulate simulates a discrete ssm with given inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: %simulate simulates a discrete ssm with given inputs

CALL: [ao_out] = simulate(sys, plist_inputs)

INPUTS:
- sys, (array of) ssm object
- plist_inputs contains 9 fields :
  - 2 noise fields
    - 'noise variable names' with a cell vector of strings of
      the desired input variable names
    - 'covariance' which gives the covariance of this noise between
      the different corresponding inputs given for the *TIME
      CONTINUOUS* noise model
  - 2 fields for input aos :
    - 'aos variable names' with a cell vector of strings of
      the desired input variable names
    - 'mean' which gives the aos for the different
      corresponding inputs
  - 2 fields to plug in constants :
    - 'constant variable names' with a cell vector of strings of
      the desired input variable names
    - 'constants' which gives the DC value for the different
      corresponding inputs
  - 2 arrays telling which blocks should be stored and returned
    - 'return states' : for blocks of the state space (ex [8])
    - 'return outputs' : for blocks of the output (ex [1 5])
  - 1 field 'Nsamples' telling max number of samples to use
    (it is smaller if aos are shorter than Nsamples)
  - 1 field 'ssini' is a cell array of vectors that give the
    initial position for simulation. 0 is the default.

***      the plist CANNOT be arrays anymore.      ***
Because of the very large number of output it is not desirable to have
more than 1 simulation per function run.

M-FILE INFO: Get information about this methods by calling
>> ssm.getInfo('simulate')

            Get information about a specified set-plist by calling:
            >> ssm.getInfo('simulate', 'Default')

***** THERE ARE NO DEFAULT PARAMETERS *****

VERSION: $Id:$

HISTORY:
04-08-2008 A Grynagier : new multiple state spaces, independant variables
in aos...
12-06-2008 M Weyrich : change for multiple outputs, bug fixes
17-04-2008 A Grynagier
19-02-2008 A Grynagier
17-02-2008 A Grynagier
Creation 30-01-2008 M Weyrich

TO DO: Check input aos for the timestep, tsdata, and ssm.timestep
options to be defined (NL case)
add check if one input mach no ssm input variable
allow use of other LTPDA functions to generate white noise
```



CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 `ssm` SSM statespace model class constructor.

SUBFUNCTIONS ^

-  `function ii = getInfo(varargin)`
-  `function plo = getDefaultPlist(set)`

SOURCE CODE ^

```
0001 %simulate simulates a discrete ssm with given inputs
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: %simulate simulates a discrete ssm with given inputs
0005 %
0006 % CALL: [ao_out] = simulate(sys, plist_inputs)
0007 %
0008 % INPUTS:
0009 %     - sys, (array of) ssm object
0010 %     - plist_inputs contains 9 fields :
0011 %         - 2 noise fields
0012 %             - 'noise variable names' with a cell vector of strings of
0013 %               the desired input variable names
0014 %             - 'covariance' which gives the covariance of this noise between
0015 %               the different corresponding inputs given for the *TIME
0016 %               CONTINUOUS* noise model
0017 %         - 2 fields for input aos :
0018 %             - 'aos variable names' with a cell vector of strings of
0019 %               the desired input variable names
0020 %             - 'mean' which gives the aos for the different
0021 %               corresponding inputs
0022 %         - 2 fields to plug in constants :
0023 %             - 'constant variable names' with a cell vector of strings of
0024 %               the desired input variable names
0025 %             - 'constants' which gives the DC value for the different
0026 %               corresponding inputs
0027 %         - 2 arrays telling which blocks should be stored and returned
0028 %             - 'return states' : for blocks of the state space (ex [8])
0029 %             - 'return outputs' : for blocks of the output (ex [1 5])
0030 %         - 1 field 'Nsamples' telling max number of samples to use
0031 %             (it is smaller if aos are shorter than Nsamples)
0032 %         - 1 field 'ssini' is a cell array of vectors that give the
0033 %             initial position for simulation. 0 is the default.
0034 %
0035 %     *** the plist CANNOT be arrays anymore. ***
0036 %     Because of the very large number of output it is not desirable to have
0037 %     more than 1 simulation per function run.
0038 %
0039 %
0040 % M-FILE INFO: Get information about this methods by calling
0041 %               >> ssm.getInfo('simulate')
0042 %
0043 %               Get information about a specified set-plist by calling:
0044 %               >> ssm.getInfo('simulate', 'Default')
0045 %
0046 % ***** THERE ARE NO DEFAULT PARAMETERS *****
0047 %
0048 % VERSION: $Id:$
0049 %
0050 % HISTORY:
0051 % 04-08-2008 A Grynagier : new multiple state spaces, independant variables
0052 % in aos...
0053 % 12-06-2008 M Weyrich : change for multiple outputs, bug fixes
0054 % 17-04-2008 A Grynagier
0055 % 19-02-2008 A Grynagier
0056 % 17-02-2008 A Grynagier
0057 % Creation 30-01-2008 M Weyrich
0058 %
0059 % TO DO: Check input aos for the timestep, tsdata, and ssm.timestep
```

```

0060 % options to be defined (NL case)
0061 % add check if one input mach no ssm input variable
0062 % allow use of other LTPDA functions to generate white noise
0063 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064
0065 function varargout = simulate(varargin)
0066
0067 %% starting initial checks
0068 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0069
0070 % Check if this is a call for parameters
0071 if utils.helper.isinfocall(varargin{:})
0072     varargout{1} = getInfo(varargin{3});
0073     return
0074 end
0075
0076 % checking number of inputs work data
0077 if ~nargin==2, error('wrong number of inputs!'), end
0078
0079 if ~isequal(class(varargin{1}), 'ssm'), error(['argument is not a ssm but a ',
class(varargin{1})]), end
0080 if ~isequal(class(varargin{2}), 'plist'), error(['argument is not a plist but a ',
class(varargin{2})]), end
0081
0082 sys = varargin{1};
0083 plist_inputs = combine(varargin{2}, getDefaultPlist('Default'));
0084 %% begin function body
0085
0086 i_sys=1;
0087 % for i_sys = 1:Nsys
0088 %% retrieve system infos
0089 if ~sys(i_sys).isnumerical
0090     error(['error because system ', sys(i_sys).name, ' is not numerical']);
0091 end
0092 M2 = ssm.cell_fusion(sys(i_sys).mmats, sys(i_sys).sssizes, sys(i_sys).sssizes);
0093 Minv = ssm.cell_recut(M2^-1, sys(i_sys).sssizes, sys(i_sys).sssizes);
0094 if ~isequal(M2, eye(size(M2)))
0095     utils.helper.msg(utils.const.msg.MNAME, 'warning, M is not the identity!!! It is:');
0096 end
0097 sssizes      = sys(i_sys).sssizes;
0098 timestep     = sys(i_sys).timestep;
0099 Ninputs      = sys(i_sys).Ninputs;
0100 Noutputs     = sys(i_sys).Noutputs;
0101 Nss          = sys(i_sys).Nss;
0102 outputsizes  = sys(i_sys).outputsizes;
0103 inputsizes   = sys(i_sys).inputsizes;
0104
0105 ssini = find(plist_inputs, 'ssini');
0106 if isempty(ssini)
0107     ssini = cell(sys(i_sys).Nss, 1);
0108     for i=1:sys(i_sys).Nss
0109         ssini{i} = zeros(sys(i_sys).sssizes(i), 1);
0110     end
0111 end
0112
0113 %% finding input and variable positions of data in option plists
0114 noise_varnames = find(plist_inputs, 'noise variable names');
0115 Nnoise = length(noise_varnames);
0116 noise_in = find(plist_inputs, 'variance');
0117 noise_pos = zeros(Nnoise, 2);
0118
0119 aos_varnames = find(plist_inputs, 'aos variable names');
0120 Naos_in = length(aos_varnames);
0121 aos_in = find(plist_inputs, 'aos');
0122 aos_pos = zeros(size(aos_varnames), 2);
0123
0124 constants_varnames = find(plist_inputs, 'constant variable names');
0125 Nconstants = length(constants_varnames);
0126 constants_in = find(plist_inputs, 'constants');
0127 constants_pos = zeros(size(constants_varnames), 2);
0128
0129 for i=1:Ninputs
0130     for j=1:inputsizes(i)
0131         % getting position
0132         [positions, sum, value] =
ssm.cellstrfind(noise_varnames, sys(i_sys).inputvarnames{i}{j}, 'all');
0133         % checking against redundant input data
0134         if sum==1
0135             noise_pos(positions, :) = [i, j];
0136         elseif sum>1
0137             error(['more than one noise variable is named ', value]);
0138         end
0139         % getting position
0140         [positions, sum, value] =
ssm.cellstrfind(aos_varnames, sys(i_sys).inputvarnames{i}{j}, 'all');
0141         % checking against redundant input data

```

```

0142         if sum==1
0143             aos_pos(positions,:) = [i,j];
0144             aos_data{positions} = aos_in(positions).data.getY ;
0145         elseif sum>1
0146             error(['more than one so variable is named ', value]);
0147         end
0148         % getting position
0149         [positions, sum, value] =
ssm.cellstrfind(constants_varnames, sys(i_sys).inputvarnames{i}{j}, 'all');
0150         % checking against redundant input data
0151         if sum==1
0152             constants_pos(positions,:) = [i,j];
0153         elseif sum>1
0154             error(['more than one constant variable is named ', value]);
0155         end
0156     end
0157 end
0158 %% getting correct number of samples
0159 Nsamples = find(plist_inputs, 'Nsamples');
0160 f0 = 1/timestep;
0161 for i=1:Naos_in
0162     Nsamples = min(Nsamples, length(aos_in(i).data.y));
0163     if ~(f0==aos_in(i).data.fs)
0164         str = ['WARNING : ssm frequency is ', num2str(f0), ...
0165             ' but sampling frequency of ao named ' ...
0166             aos_in(i).name, ' is ', num2str(aos_in(i).data.fs) ];
0167         utils.helper.msg(utils.const.msg.MNAME, str);
0168     end
0169     % maybe tdata should be retrieved and verified to be equal, rather than this.
0170 end
0171 if Nsamples == inf % case there is no input!
0172     Nsamples = 0;
0173 end
0174
0175 %% generating random data
0176 [V,E]=eig(noise_in);
0177 noise_mat = (V*(E'.^0.5)*V');
0178 noise_array = noise_mat*randn(Nnoise, Nsamples);
0179 % How to generate initial position
0180 % Do one step with firts noise vector
0181 % multiply inputs by (I - Minv*A)^-1
0182 % add in the fashion
0183 %
0184 %% evaluating data to return
0185 return_states = find(plist_inputs, 'return states');
0186 returnStates = false(1,Nss);
0187 for i=1:Nss
0188     returnStates(i) = ~isempty(find( ismember(return_states, i)));
0189 end
0190 return_outputs = find(plist_inputs, 'return outputs');
0191 returnOutputs = false(1, Noutputs);
0192 for i=1:Noutputs
0193     returnOutputs(i) = ~isempty(find( ismember(return_outputs, i)));
0194 end
0195 %% simulation first step
0196 x = cell(Nss, 1);
0197 lastX = ssini;
0198 for i=1:Nss
0199     if returnStates(i)
0200         x{i} = zeros(sssizes(i), Nsamples);
0201     end
0202 end
0203 y = cell(Noutputs, 1);
0204 for i=1:Noutputs
0205     if returnOutputs(i)
0206         y{i} = zeros(outputsizes(i), Nsamples);
0207     end
0208 end
0209 IncrementX = cell(Nss,1);
0210 IncrementY = cell(Noutputs,1);
0211 %% simulation loop
0212 for k = 1:Nsamples
0213     if rem(k,500)==0
0214         utils.helper.msg(utils.const.msg.MNAME, ['simulation time : ', num2str(k*timestep)
0215         ]);
0216     end
0217     %% resetting each field
0218     for j = 1:Nss
0219         IncrementX{j} = zeros(sssizes(j),1);
0220     end
0221     for j = 1:Noutputs
0222         IncrementY{j} = zeros(outputsizes(j),1);
0223     end
0224     %% evaluating differences
0225     for i=1:Nnoise %Noise MUST BE MODIFIED IF IT IS REALLY SPECIFIED FOR TIME
    CONTINUOUS SYSTEM!!!!!!

```



```

0225     for s=1:Nss
0226         if ~ isempty(sys(i_sys).bmats{s,noise_pos(i,1)})
0227             IncrementX{s} = IncrementX{s} +
sys(i_sys).bmats{s,noise_pos(i,1)}(:,noise_pos(i,2))*noise_array(i,k);
0228         end
0229     end
0230     for j = 1:Noutputs
0231         if ~ isempty(sys(i_sys).dmats{j,noise_pos(i,1)})
0232             IncrementY{j} = IncrementY{j} +
sys(i_sys).dmats{j,noise_pos(i,1)}(:,noise_pos(i,2))*noise_array(i,k);
0233         end
0234     end
0235 end
0236
0237     for i=1:Naos_in
0238         for s=1:Nss
0239             if ~ isempty(sys(i_sys).bmats{s,aos_pos(i,1)})
0240                 IncrementX{s} = IncrementX{s} +
sys(i_sys).bmats{s,aos_pos(i,1)}(:,aos_pos(i,2))*aos_data{i}(k);
0241             end
0242         end
0243         for j = 1:Noutputs
0244             if ~ isempty(sys(i_sys).dmats{j,aos_pos(i,1)})
0245                 IncrementY{j} = IncrementY{j} +
sys(i_sys).dmats{j,aos_pos(i,1)}(:,aos_pos(i,2))*aos_data{i}(k);
0246             end
0247         end
0248     end
0249
0250     for i=1:Nconstants
0251         for s=1:Nss
0252             if ~ isempty(sys(i_sys).bmats{s,constants_pos(i,1)})
0253                 IncrementX{s} = IncrementX{s} +
sys(i_sys).bmats{s,constants_pos(i,1)}(:,constants_pos(i,2))*constants_in(i);
0254             end
0255         end
0256         for j = 1:Noutputs
0257             if ~ isempty(sys(i_sys).dmats{j,constants_pos(i,1)})
0258                 IncrementY{j} = IncrementY{j} +
sys(i_sys).dmats{j,constants_pos(i,1)}(:,constants_pos(i,2))*constants_in(i);
0259             end
0260         end
0261     end
0262     %% computing and storing outputs
0263     %% differential equation
0264     lastX = ssm.cell_mult(Minv, ssm.cell_add(IncrementX, ssm.cell_mult(sys(i_sys).amats,
lastX)));
0265     %% output equations
0266     lastY = ssm.cell_add(ssm.cell_mult(sys(i_sys).cmats, lastX) ,IncrementY);
0267     for s=1:Nss
0268         if returnStates(s)
0269             x{s}(:,k) = lastX{s};
0270         end
0271     end
0272     for o=1:Noutputs
0273         if returnOutputs(o)
0274             y{o}(:,k) = lastY{o};
0275         end
0276     end
0277 end
0278 %% saving in aos
0279 Naos = 0;
0280 for i=1:Nss
0281     if returnStates(i)
0282         Naos = Naos +sssizes(i);
0283     end
0284 end
0285 for i=1:Noutputs
0286     if returnOutputs(i)
0287         Naos = Naos +outputsizes(i);
0288     end
0289 end
0290 if Naos>0
0291     ao_out(Naos) = ao;
0292     if ~exist('t','var')
0293         t = (1:Nsamples)*timestep;
0294     end
0295     count = 1;
0296     for i=1:Nss
0297         if returnStates(i)
0298             for j=1:sssizes(i)
0299                 ao_out(count) = ao(tsdata(t,x{i}(j,:)));
0300                 ao_out(count).setName(['sim_X_',num2str(i),'_',num2str(j)]);
0301                 ao_out(count).setDescription(...
0302                     ['ssm#:',num2str(i_sys),...
0303                     ', state:',sys(i_sys).ssnames{i},' ',sys(i_sys).ssvarnames{i}{j}]);

```

```

0304         ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs
, {''}, sys(i_sys).hist );
0305         count = count+1;
0306     end
0307 end
0308 end
0309 for i=1:Noutputs
0310     if returnOutputs(i)
0311         for j=1:outputsizes(i)
0312             ao_out(count) = ao(tsddata(t,y{i}(j,:)));
0313             ao_out(count).setName(['sim_Y_', num2str(i), '_', num2str(j)]);
0314             ao_out(count).setDescription( ...
0315                 ['ssm: ', num2str(i_sys), ...
0316                 ', output: ', sys(i_sys).outputnames{i}, ' variable:
', sys(i_sys).outputvarnames{i}{j}]);
0317             ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs
, {''}, sys(i_sys).hist );
0318             count = count+1;
0319         end
0320     end
0321 end
0322 end
0323 %% construct output analysis object
0324 plist_out = plist('lastX', lastX);
0325 varargout = {ao_out plist_out};
0326 end
0327
0328 function ii = getInfo(varargin)
0329     if nargin == 1 && strcmpi(varargin{1}, 'None')
0330         sets = {};
0331         pls = [];
0332     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0333         sets{1} = varargin{1};
0334         pls = getDefaultPlist(sets{1});
0335     else
0336         sets = {'Default'};
0337         pls = [];
0338         for kk=1:numel(sets)
0339             pls = [pls getDefaultPlist(sets{kk})];
0340         end
0341     end
0342     % Build info object
0343     ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38
ingo Exp $', sets, pls);
0344 end
0345
0346 function plo = getDefaultPlist(set)
0347     switch set
0348     case 'Default'
0349         plo = plist(...
0350             'noise variable names', {}, ...
0351             'variance', {}, ...
0352             'aos variable names', {}, ...
0353             'aos', {}, ...
0354             'constant variable names', {}, ...
0355             'constant', {}, ...
0356             'return states', [], ...
0357             'return outputs', 1, ...
0358             'Nsamples', 1e4, ...
0359             'ssini', {} ...
0360         );
0361     otherwise
0362         error('### Unknown parameter set [%s].', set);
0363     end
0364 end

```


ssm2iirpz

PURPOSE ^

ssm2iirpz converts a statespace model object to an miir or a pzmodel

SYNOPSIS ^

function varargout = ssm2iirpz(varargin)

DESCRIPTION ^

```
ssm2iirpz converts a statespace model object to an miir or a pzmodel
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: ssm2iirpz converts a statespace model object to an miir or a
             pzmodel object.

CALL:
    >> varargout = ssm2iirpz(ssm, options);

INPUT :
    ssm      - a ssm object
    options  - a plist with parameters 'inputs', 'states' and
               'outputs' to indicate which inputs, states and output
               variables are taken in account. This requires proper
               variable naming. If a variable called appears more
               that once it will be used once only.

               The field may be :
               - a cellstr containing the resp. input/state/output *variable* names
               - a logical indexing the resp. input/state/output *variables*
                 names. Index is stored in a cell array, each cell
                 corresponding to one input/state/output block.
               - a double indexing the resp. input/state/output *variables*
                 names. Index is stored in a cell array, each cell
                 corresponding to one input/state/output block.
               - 'ALL', this string indicates all i/o variables will be
                 given

OUTPUT:
    - an array of miir object if the timestep is greater than zero
    - an array of pzm object if the timestep is zero

M-FILE INFO: Get information about this methods by calling
    >> ssm.getInfo('ssm2iirpz')

               Get information about a specified set-plist by calling:
    >> ssm.getInfo('ssm2iirpz', 'Default')

VERSION:      $Id: ssm2ss.m,v 1.3 2008/04/21 12:38:57 adrien Exp $

HISTORY:      21-04-2008 A Grynagier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY Make copy of ssm objects depending of the second input
-  **double** Convert a statespace model object to double arrays for given i/o
-  **minreal** minreal gives a minimal realization of a ssm object by deleting state

This function is called by:

 **ssm** SSM statespace model class constructor.

SUBFUNCTIONS ^

 function varargout = roots2poly(roots)
 function ii = getInfo(varargin)
 function plo = getDefaultPlist(set)

SOURCE CODE ^

```

0001 % ssm2iirpz converts a statespace model object to an miir or a pzmodel
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ssm2iirpz converts a statespace model object to an miir or a
0005 %               pzmodel object.
0006 %
0007 % CALL:
0008 %               >> varargout = ssm2iirpz(ssm, options);
0009 %
0010 % INPUT :
0011 %
0012 %       ssm      - a ssm object
0013 %       options  - a plist with parameters 'inputs', 'states' and
0014 %                 'outputs' to indicate which inputs, states and output
0015 %                 variables are taken in account. This requires proper
0016 %                 variable naming. If a variable called appears more
0017 %                 that once it will be used once only.
0018 %
0019 %       The field may be :
0020 %       - a cellstr containing the resp. input/state/output *variable* names
0021 %       - a logical indexing the resp. input/state/output *variables*
0022 %       names. Index is stored in a cell array, each cell
0023 %       corresponding to one input/state/output block.
0024 %       - a double indexing the resp. input/state/output *variables*
0025 %       names. Index is stored in a cell array, each cell
0026 %       corresponding to one input/state/output block.
0027 %       - 'ALL', this string indicates all i/o variables will be
0028 %       given
0029 %
0030 % OUTPUT:
0031 %
0032 % - an array of miir object if the timestep is greater than zero
0033 % - an array of pzm object if the timestep is zero
0034 %
0035 % M-FILE INFO: Get information about this methods by calling
0036 %               >> ssm.getInfo('ssm2iirpz')
0037 %
0038 %               Get information about a specified set-plist by calling:
0039 %               >> ssm.getInfo('ssm2iirpz', 'Default')
0040 %
0041 % VERSION:      $Id: ssm2ss.m,v 1.3 2008/04/21 12:38:57 adrien Exp $
0042 %
0043 % HISTORY: 21-04-2008 A Grynagier
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046
0047 function varargout = ssm2iirpz(varargin)
0048
0049 %% starting initial checks
0050 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0051
0052 % Check if this is a call for parameters
0053 if utils.helper.isinfocall(varargin{:})
0054     varargout{1} = getInfo(varargin{3});
0055     return
0056 end
0057
0058 % Collect input variable names
0059 in_names = cell(size(varargin));
0060 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0061
0062 % Collect all SSMs and plists
0063 [sys, ssm_invars] = utils.helper.collect_objects(varargin(:), 'ssm', in_names);
0064 pl                = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0065
0066 options = combine(pl, getDefaultPlist('Default'));
0067
0068 if numel(sys) ~= 1
0069     error('### Input (only) one SSM object.');
```

```

0070 end
0071
0072 % Decide on a deep copy or a modify, depending on the output
0073 sys = copy(sys, nargin);
0074
0075 %% begin function body
0076
0077 %% convert to double
0078 % Convert to double arrays
0079 [A,B,C,D,Ts,inputvarnames,ssvarnames,outputvarnames] = double(sys, options);
0080
0081
0082 Ninputs_out = numel(inputvarnames);
0083 Noutputs_out = numel(outputvarnames);
0084
0085 if Ts > 0
0086     %% convert to miir
0087     miir_out(Noutputs_out, Ninputs_out) = miir();
0088     for i_inputs=1:Ninputs_out
0089         for i_outputs=1:Noutputs_out
0090             sys_loc = ss( A,B(:,i_inputs),C(i_outputs,:),D(i_outputs, i_inputs ),Ts);
0091             sys_loc = minreal(sys_loc);
0092             [A_loc, B_loc, C_loc, D_loc]=ssdata(sys_loc);
0093             [a,b] = ss2tf(A_loc, B_loc, C_loc, D_loc);
0094             name = [ssm_invars{1}, ' : ', inputvarnames{i_inputs}, ' -> ',
0095                 outputvarnames{i_outputs}];
0096             if b==0
0097                 m = miir();
0098             else
0099                 m = miir(real(a), real(b), 1/sys.timestep);
0100             end
0101             m.addHistory(ssm.getInfo(mfilename), options , ssm_invars, sys.hist );
0102             m.name = name;
0103             miir_out(i_outputs, i_inputs) = m;
0104         end
0105     end
0106     varargout = {miir_out};
0107 else
0108     %% convert to pzm
0109     pzm_out(Noutputs_out, Ninputs_out) = pzmodel();
0110     for i_inputs=1:Ninputs_out
0111         for i_outputs=1:Noutputs_out
0112             [b,a] = ss2tf(A,B(:,i_inputs),C(i_outputs,:),D(i_inputs, i_outputs));
0113             p = roots(b);
0114             p = roots2poly(p);
0115             z = roots(a);
0116             z = roots2poly(z);
0117             Gain = a(1)/b(1);
0118             name = [ssm_invars, '_', num2str(i_inputs), ': ', num2str(i_outputs)];
0119             p = pzmodel(Gain,p,z,name);
0120             p.addHistory(ssm.getInfo(mfilename), options , ssm_invars, sys.hist );
0121             pzm_out(i_outputs,i_inputs) = p;
0122         end
0123     end
0124     varargout = {pzm_out};
0125 end
0126
0127
0128
0129 function varargout = roots2poly(roots)
0130 poly = pz();
0131 i=1;
0132 while i < length(roots)+1
0133     if isreal(roots(i))
0134         poly = [poly pz(roots(i))]; %#ok<AGROW>
0135         i = i+1;
0136     else
0137         f = norm(roots(i))*sign(real(roots(i)));
0138         Q = norm(roots(i))/abs(2*real(roots(i)));
0139         poly = [poly pz(f,Q)]; %#ok<AGROW>
0140         i = i+2; %#ok<FXSET>
0141     end
0142 end
0143 varargout = {poly};
0144 end
0145
0146
0147 %-----
0148 % Get Info Object
0149 %-----
0150 function ii = getInfo(varargin)
0151 if nargin == 1 && strcmpi(varargin{1}, 'None')
0152     sets = {};
0153     pls = [];
0154 elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})

```

```
0155     sets{1} = varargin{1};
0156     pls = getDefaultPlist(sets{1});
0157 else
0158     sets = {'Default'};
0159     pls = [];
0160     for kk=1:numel(sets)
0161         pls = [pls getDefaultPlist(sets{kk})];
0162     end
0163 end
0164 % Build info object
0165 ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v
1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
0166 end
0167
0168 %-----
0169 % Get Default Plist
0170 %-----
0171 function plo = getDefaultPlist(set)
0172     switch set
0173     case 'Default'
0174         plo = ssm.getInfo('reduce_model', 'Default').plists;
0175     otherwise
0176         error('### Unknown parameter set [%s].', set);
0177     end
0178 end
0179
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ssm2ss

PURPOSE ^

SSM2SS converts a statespace model object to a MATLAB statespace object.

SYNOPSIS ^

function varargout = ssm2ss(varargin)

DESCRIPTION ^

```
SSM2SS converts a statespace model object to a MATLAB statespace object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SSM2SS converts a statespace model object to a MATLAB statespace object.

CALL:      Convert a statespace model object to a MATLAB statespace object.
>> ss = ssm2ss(ssm, options);

ssm -      ssm object
plist with parameters 'inputs', 'states' and 'outputs' to indicate which
inputs, states and outputs variables are taken in account. This requires
proper variable naming. If a variable called appears more that once it
will be used once only.
The field may be :
    - a cellstr containing the resp. input/state/output *variable* names
    - a logical indexing the resp. input/state/output *variables*
      names. Index is stored in a cell array, each cell
      correponding to one input/state/output block.
    - a double indexing the resp. input/state/output *variables*
      names. Index is stored in a cell array, each cell
      correponding to one input/state/output block.
    - 'ALL', this string indicates all i/o variables will be
      given

VERSION:    $Id: ssm2ss.m,v 1.18 2008/08/26 20:55:27 adrien Exp $

M-FILE INFO: Get information about this methods by calling
>> ssm.getInfo('ssm2ss')

              Get information about a specified set-plist by calling:
>> ssm.getInfo('ssm2ss', 'Default')

HISTORY: 16-04-08 M Hewitson
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [double](#) Convert a statespace model object to double arrays for given i/o

This function is called by:

-  [minreal](#) minreal gives a minimal realization of a ssm object by deleting state
-  [ssm](#) SSM statespace model class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```

0001 % SSM2SS converts a statespace model object to a MATLAB statespace object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SSM2SS converts a statespace model object to a MATLAB statespace object.
0005 %
0006 % CALL:          Convert a statespace model object to a MATLAB statespace object.
0007 %               >> ss = ssm2ss(ssm, options);
0008 %
0009 % ssm -          ssm object
0010 % plist with parameters 'inputs', 'states' and 'outputs' to indicate which
0011 % inputs, states and outputs variables are taken in account. This requires
0012 % proper variable naming. If a variable called appears more than once it
0013 % will be used once only.
0014 % The field may be :
0015 %               - a cellstr containing the resp. input/state/output *variable* names
0016 %               - a logical indexing the resp. input/state/output *variables*
0017 %               names. Index is stored in a cell array, each cell
0018 %               corresponding to one input/state/output block.
0019 %               - a double indexing the resp. input/state/output *variables*
0020 %               names. Index is stored in a cell array, each cell
0021 %               corresponding to one input/state/output block.
0022 %               - 'ALL', this string indicates all i/o variables will be
0023 %               given
0024 %
0025 % VERSION:       $Id: ssm2ss.m,v 1.18 2008/08/26 20:55:27 adrien Exp $
0026 %
0027 % M-FILE INFO:   Get information about this methods by calling
0028 %               >> ssm.getInfo('ssm2ss')
0029 %
0030 %               Get information about a specified set-plist by calling:
0031 %               >> ssm.getInfo('ssm2ss', 'Default')
0032 %
0033 % HISTORY: 16-04-08 M Hewitson
0034 %           Creation
0035 %
0036 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0037
0038 function varargout = ssm2ss(varargin)
0039
0040 %% starting initial checks
0041 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0042
0043 % Check if this is a call for parameters
0044 if utils.helper.isinfocall(varargin{:})
0045     varargout{1} = getInfo(varargin{3});
0046     return
0047 end
0048
0049 % checking number of inputs work data
0050 if ~nargin==2, error('wrong number of inputs!'), end
0051
0052 if ~isequal(class(varargin{1}), 'ssm'), error(['argument is not a ssm but a ',
class(varargin{1})]), end
0053 if ~isequal(class(varargin{2}), 'plist'), error(['argument is not a plist but a ',
class(varargin{2})]), end
0054
0055 obj = varargin{1};
0056 options = combine(varargin{2}, getDefaultPlist('Default'));
0057
0058 %% begin function body
0059 if ~obj.isnumerical
0060     error('system should be numeric')
0061 end
0062
0063 [A,B,C,D,Ts,inputvarnames,ssvarnames, outputvarnames] = double(obj, options);
0064 sys = ss(A,B,C,D,Ts);
0065 sys.StateName = ssvarnames;
0066 sys.InputName = inputvarnames;
0067 sys.OutputName = outputvarnames;
0068 sys.Notes = obj.description;
0069 sys.Name = obj.name;
0070 varargout{1} = sys;
0071
0072 end
0073
0074
0075 function ii = getInfo(varargin)
0076     if nargin == 1 && strcmpi(varargin{1}, 'None')
0077         sets = {};
0078         pls = [];
0079     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0080         sets{1} = varargin{1};

```

```
0081     pls = getDefaultPlist(sets{1});
0082 else
0083     sets = {'Default'};
0084     pls = [];
0085     for kk=1:numel(sets)
0086         pls = [pls getDefaultPlist(sets{kk})];
0087     end
0088 end
0089 % Build info object
0090 ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: ssm2ss.m,v 1.18 2008/08/26
20:55:27 adrien Exp $', sets, pls);
0091 end
0092
0093 function plo = getDefaultPlist(set)
0094     switch set
0095     case 'Default'
0096         plo = ssm.getInfo('reduce_model', 'Default').plists;
0097     otherwise
0098         error('### Unknown parameter set [%s].', set);
0099     end
0100 end
0101
```

mfir

PURPOSE ^

MFIR FIR filter object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
MFIR FIR filter object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MFIR FIR filter object class constructor.
              Create a mfir object.

SUPER CLASSES: ltpda_filter < ltpda_uoh < ltpda_uo < ltpda_obj

PROPERTIES:

  Inherit Properties (read only)
    name      - name of object
    created   - creation time (time-object)
    prov      - contains a instance of the provenance class.
    hist      - history of the object (history object)
    version   - cvs-version string.
    fs        - sample rate that the filter is designed for
    infile    - filename if the filter was loaded from file
    a         - set of numerator coefficients
    histout   - output history values to filter

  Protected Properties (read only)
    ntps      - number of coefficients in the filter
    gd        -

MFIR Methods:

  Defined Abstract methods:
    char      - returns one character string which represents the object
    copy      - copies an object
    display   - displays an object
    string    - converts an object to a command string which will
                recreate the plist object.
    update_struct - updates a object structure to the current tbx-version

  Public methods:
    redesign  - redesign the input filter to work for the given sample rate.
    resp      - make a frequency response of the filter.

  Protected methods:
    setGd     - set the property 'gd'

  Private methods:
    fromFile  - construct an mfir filter from a file
    fromRepository - construct an mfir filter from a repository
    fromA     - construct an mfir from coefficients
    fromAO    - create FIR filter from magnitude of input AO/fsdata
    fromPzmodel - construct an mfir from a pzmodel
    fromStandard - construct an mfir from a standard types
    mklowpass - return a low pass filter
    mkhighpass - return a high pass filter
    mkbandpass - return a bandpass filter
    mkbandreject - return a low pass filter

CONSTRUCTORS:

  f = mfir()      - creates an empty mfir object.
  f = mfir(fi)    - creates a copy of the input mfir object, fi.
  f = mfir(a)     - creates an mfir object based on the magnitude of
                    the input AO/fsdata object a.
  f = mfir(c,fs)  - creates an mfir object based on the vector of input
                    coefficients c.
```



```

                                The sample rate for which the filter is designed
                                should be specified as well.
f = mfir(filename) - creates an mfir object loading the mfir object from disk
f = mfir(pl)       - creates an mfir object from the description given
                                in the parameter list.

```

Parameter sets for plist constructor (in order of priority):

From XML File

Construct an MFIR by loading it from an XML file.

```

'filename' - construct an MFIR from a filename.
             Example: plist('filename', 'm1.xml')
             [default: empty string]

```

From MAT File

Construct an MFIR by loading it from a MAT file.

```

'filename' - construct an MFIR from a filename.
             Example: plist('filename', 'm1.mat')
             [default: empty string]

```

From Repository

Construct an MFIR by retrieving it from an LTPDA repository.

```

'Hostname' - the repository hostname. Only those objects which
             are MFIRs are returned.
             [default: 'localhost'];

```

Additional parameters:

```

'Database' - The database name [default: 'ltpda']
'ID'        - A vector of object IDs. [default: []]
'CID'       - Retrieve all MFIR objects from a particular
               collection.

```

From Standard Type

Construct an MFIR of a standard type.

```

'type' - one of the types: 'highpass', 'lowpass',
                          'bandpass', 'bandreject'
             [default: 'lowpass']

```

You can also specify optional parameters:

```

'gain' - the gain of the filter [default: 1]
'fc'    - the roll-off frequency [default: 0.1 Hz]
'fs'    - the sampling frequency to design for [default: 1 Hz]
'order' - the filter order [default: 64]
'win'   - Specify window function used in filter
             design [default: 'Hamming']

```

From AO

Construct an MFIR based on the magnitude of the input AO/fsdata object a

```

'method' - the design method:
            'frequency-sampling' - uses fir2()
            'least-squares'       - uses firls()
            'Parks-McClellan'     - uses firpm()
            [default: 'frequency-sampling']
'Win'     - Window function for frequency-sampling method
            [default: 'Hanning']
'N'       - filter order [default: 512]

```

From Plist

```

'Plist' - construct from a plist. The value passed should be a plist
          object.
          [default: empty plist]

```

EXAMPLE 1: Create an order 1 highpass filter with high frequency gain 2. Filter is designed for 10 Hz sampled data and has a cut-off frequency of 0.2 Hz.

```
>> pl = plist('type', 'highpass', ...
```

NOTES:

$$b(1)*y(n) = a(1)*x(n) + a(2)*x(n-1) + \dots + a(na+1)*x(n-na) - b(2)*y(n-1) - \dots - b(nb+1)*y(n-nb)$$

```
>> info = mfirm.getInfo(method, set)
e.g. >> info = mfirm.getInfo('mfirm', 'From Pzmodel')
```

```
HISTORY:      09-02-2007 M Hewitson
               Creation
              11-02-2008 M Hueller
               Help fixed
               Default parameter list defined
```

[illegible]

This function calls:

-  **char** CHAR convert a mfir object into a string.
-  **copy** COPY Make copy of mfir objects depending of the second input
-  **display** DISPLAY overloads display functionality for mfir objects.
-  **fromA** Construct an mfir from coefficients
-  **fromAO** create FIR filter from magnitude of input AO/fsdata
-  **fromPzmodel** Construct an mfir from a pzmodel
-  **fromStandard** Construct an mfir from a standard types
-  **mfir** MFIR FIR filter object class constructor.
-  **mkbandpass** MKBANDPASS return a bandpass filter mfir(). A Cheby filter is used.
-  **mkbandreject** MKBANDREJECT return a low pass filter mfir(). A Butterworth filter is used.
-  **mkhighpass** MKHIGHPASS return a high pass filter mfir(). A Butterworth filter is used.
-  **mklowpass** MKLOWPASS return a low pass filter mfir().
-  **parseFilterParams** PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard FIR filter.
-  **setGd** SETGD Set the property 'gd'
-  **string** STRING writes a command string that can be used to recreate the input filter object.
-  **update struct** UPDATE STRUCT update the input structure to the current ltpda version

This function is called by:

-  **copy** COPY Make copy of mfir objects depending of the second input
-  **mfir** MFIR FIR filter object class constructor.
-  **redesign** REDESIGN redesign the input filter to work for the given sample rate.

SUBFUNCTIONS ^

- function obj = set.gd(obj, val)
- function obj = set.ntaps(obj, val)
- function val = get.ntaps(obj)
- function obj = mfir(varargin)
- function out = VEROUT()
- function ii = getInfo(varargin)
- function out = SETS()
- function out = getDefaultPlist(set)

SOURCE CODE ^

```
0001 % MFIR FIR filter object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MFIR FIR filter object class constructor.
0005 %             Create a mfir object.
0006 %
0007 % SUPER CLASSES: ltpda_filter < ltpda_uoh < ltpda_uo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         name      - name of object
0013 %         created    - creation time (time-object)
0014 %         prov       - contains a instance of the provenance class.
0015 %         hist       - history of the object (history object)
0016 %         version    - cvs-version string.
0017 %         fs         - sample rate that the filter is designed for
0018 %         infile     - filename if the filter was loaded from file
0019 %         a          - set of numerator coefficients
0020 %         histout    - output history values to filter
0021 %
0022 %     Protected Properties (read only)
0023 %         ntaps      - number of coefficients in the filter
0024 %         gd         -
0025 %
0026 % MFIR Methods:
0027 %
0028 %     Defined Abstract methods:
0029 %         char        - returns one character string which represents the object
0030 %         copy        - copies an object
0031 %         display     - displays an object
0032 %         string      - converts an object to a command string which will
0033 %                     recreate the plist object.
0034 %         update_struc - updates a object structure to the current tbx-version
0035 %
0036 %     Public methods:
0037 %         redesign    - redesign the input filter to work for the given sample rate.
0038 %         resp        - make afrequency response of the filter.
0039 %
0040 %     Protected methods:
0041 %         setGd       - set the property 'gd'
0042 %
0043 %     Private methods:
0044 %         fromFile    - construct an mfir filter from a file
0045 %         fromRepository - construct an mfir filter from a repository
0046 %         fromA       - construct an mfir from coefficients
0047 %         fromAO       - create FIR filter from magnitude of input AO/fsdata
0048 %         fromPzmodel - construct an mfir from a pzmodel
0049 %         fromStandard - construct an mfir from a standard types
0050 %         mklowpass   - return a low pass filter
0051 %         mkhighpass  - return a high pass filter
0052 %         mkbandpass  - return a bandpass filter
0053 %         mkbandreject - return a low pass filter
0054 %
0055 % CONSTRUCTORS:
0056 %
0057 %         f = mfir()      - creates an empty mfir object.
0058 %         f = mfir(fi)    - creates a copy of the input mfir object, fi.
0059 %         f = mfir(a)     - creates an mfir object based on the magnitude of
0060 %                         the input AO/fsdata object a.
0061 %         f = mfir(c,fs)  - creates an mfir object based on the vector of input
0062 %                         coefficients c.
0063 %                         The sample rate for which the filter is designed
```

```

0064 % should be specified as well.
0065 % f = mfir(filename) - creates an mfir object loading the mfir object from disk
0066 % f = mfir(pl) - creates an mfir object from the description given
0067 % in the parameter list.
0068 %
0069 % Parameter sets for plist constructor (in order of priority):
0070 %
0071 % From XML File
0072 % -----
0073 %
0074 % Construct an MFIR by loading it from an XML file.
0075 %
0076 % 'filename' - construct an MFIR from a filename.
0077 % Example: plist('filename', 'm1.xml')
0078 % [default: empty string]
0079 %
0080 % From MAT File
0081 % -----
0082 %
0083 % Construct an MFIR by loading it from a MAT file.
0084 %
0085 % 'filename' - construct an MFIR from a filename.
0086 % Example: plist('filename', 'm1.mat')
0087 % [default: empty string]
0088 %
0089 % From Repository
0090 % -----
0091 %
0092 % Construct an MFIR by retrieving it from an LTPDA repository.
0093 %
0094 % 'Hostname' - the repository hostname. Only those objects which
0095 % are MFIRs are returned.
0096 % [default: 'localhost'];
0097 %
0098 % Additional parameters:
0099 %
0100 % 'Database' - The database name [default: 'ltpda']
0101 % 'ID' - A vector of object IDs. [default: []]
0102 % 'CID' - Retrieve all MFIR objects from a particular
0103 % collection.
0104 %
0105 % From Standard Type
0106 % -----
0107 %
0108 % Construct an MFIR of a standard type.
0109 %
0110 % 'type' - one of the types: 'highpass', 'lowpass',
0111 % 'bandpass', 'bandreject' [default: 'lowpass']
0112 %
0113 % You can also specify optional parameters:
0114 % 'gain' - the gain of the filter [default: 1]
0115 % 'fc' - the roll-off frequency [default: 0.1 Hz]
0116 % 'fs' - the sampling frequency to design for [default: 1 Hz]
0117 % 'order' - the filter order [default: 64]
0118 % 'win' - Specify window function used in filter
0119 % design [default: 'Hamming']
0120 %
0121 % From AO
0122 % -----
0123 %
0124 % Construct an MFIR based on the magnitude of the input AO/fsdata object a
0125 %
0126 % 'method' - the design method:
0127 % 'frequency-sampling' - uses fir2()
0128 % 'least-squares' - uses firls()
0129 % 'Parks-McClellan' - uses firpm()
0130 % [default: 'frequency-sampling']
0131 % 'Win' - Window function for frequency-sampling method
0132 % [default: 'Hanning']
0133 % 'N' - filter order [default: 512]
0134 %
0135 % From Plist
0136 % -----
0137 %
0138 % 'Plist' - construct from a plist. The value passed should be a plist
0139 % object.
0140 % [default: empty plist]
0141 %
0142 %
0143 %
0144 % EXAMPLE 1: Create an order 1 highpass filter with high frequency gain 2.
0145 % Filter is designed for 10 Hz sampled data and has a cut-off
0146 % frequency of 0.2 Hz.
0147 %
0148 % >> pl = plist('type', 'highpass', ...
0149 % 'order', 1, ...

```

```

0150 %             'gain', 2.0,      ...
0151 %             'fs',   10,      ...
0152 %             'fc',   0.2);
0153 %         >> f = mfir(pl)
0154 %
0155 % NOTES:
0156 %         ** The convention used here for naming the filter coefficients is
0157 %            the opposite to MATLAB's convention. The recursion formula
0158 %            for this convention is
0159 %
0160 %            
$$b(1)y(n) = a(1)x(n) + a(2)x(n-1) + \dots + a(na+1)x(n-na)$$

0161 %            
$$- b(2)y(n-1) - \dots - b(nb+1)y(n-nb)$$

0162 %
0163 %
0164 % M-FILE INFO: The following call returns an minfo object that contains
0165 %               information about the mfir constructor:
0166 %               >> info = mfir.getInfo
0167 %               or   >> info = mfir.getInfo('mfir')
0168 %
0169 %               You can get information about class methods by calling:
0170 %               >> info = mfir.getInfo(method)
0171 %               e.g. >> info = mfir.getInfo('eq')
0172 %
0173 %               You can also restrict the sets of parameters contained in
0174 %               the minfo object by calling:
0175 %               >> info = mfir.getInfo(method, set)
0176 %               e.g. >> info = mfir.getInfo('mfir', 'From Pzmodel')
0177 %
0178 % VERSION:      $Id: mfir.m,v 1.63 2008/09/07 10:27:46 hewitson Exp $
0179 %
0180 % HISTORY:      09-02-2007 M Hewitson
0181 %                Creation
0182 %                11-02-2008 M Hueller
0183 %                Help fixed
0184 %                Default parameter list defined
0185 %
0186 % SEE ALSO:     miir, ltpda_filter, ltpda_uoh, ltpda_uo, ltpda_obj, plist
0187 %
0188 %%%%%%%%%%%%%%%
0189 %
0190 classdef mfir < ltpda_filter
0191 %%%%%%%%%%%%%%%
0192 %%%%%%%%%%%%%%% Property definition %%%%%%%%%%%%%%%
0193 %%%%%%%%%%%%%%%
0194 %%%%%%%%%%%%%%%
0195 %----- Public (read/write) Properties -----
0196 properties
0197 end
0198 %----- Protected read-only Properties -----
0199 properties (SetAccess = protected)
0200     gd = [];
0201     version = '$Id: mfir.m,v 1.63 2008/09/07 10:27:46 hewitson Exp $';
0202 end
0203 %----- Protected read-only Properties -----
0204 properties (SetAccess = protected, Dependent = true)
0205     ntabs
0206 end
0207 %----- Private Properties -----
0208 properties (GetAccess = protected, SetAccess = protected)
0209 end
0210 %----- Abstract Properties -----
0211 properties (Abstract = true, SetAccess = protected)
0212 end
0213 %%%%%%%%%%%%%%% Check property setting %%%%%%%%%%%%%%%
0214 %%%%%%%%%%%%%%%
0215 %%%%%%%%%%%%%%%
0216 methods
0217     function obj = set.gd(obj, val)
0218         if ~isempty(val)
0219             if ~isnumeric(val) || ~isreal(val)
0220                 error('### The value for the property ''gd'' must be a real number(s)');
0221             end
0222         end
0223         obj.gd = val;
0224     end
0225     function obj = set.ntaps(obj, val)
0226         error('### Don''t set the property ''ntaps''. It is computed by length(a).');
0227     end
0228 end
0229
0230
0231
0232
0233
0234
0235

```

```

0236 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0237 % Compute the Dependent properties %
0238 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0239
0240 methods
0241     function val = get.ntaps(obj)
0242         val = length(obj.a);
0243     end
0244 end
0245
0246 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0247 % Constructor %
0248 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0249
0250 methods
0251     function obj = mfir(varargin)
0252
0253         import utils.const.*
0254         utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0255
0256         %% Call superclass
0257         obj = obj@ltpda_filter(varargin{:});
0258
0259         %%%%%%%%%%% Set default values %%%%%%%%%%%
0260
0261         % Collect all mfir objects
0262         [fs, invars, rest] = utils.helper.collect_objects(varargin(:), 'mfir');
0263
0264         if isempty(rest) && ~isempty(fs)
0265             % Do copy constructor and return
0266             utils.helper.msg(msg.OPROC1, 'copy constructor');
0267             obj = copy(fs, 1);
0268             for kk=1:numel(obj)
0269                 obj(kk).addHistory(mfir.getInfo('mfir', 'None'), [], [], obj(kk).hist);
0270             end
0271             return
0272         end
0273
0274         switch nargin
0275             case 0
0276                 %%%%%%%%%%% no inputs %%%%%%%%%%%
0277                 %%%%%%%%%%% empty constructor %%%%%%%%%%%
0278                 utils.helper.msg(msg.OPROC1, 'empty constructor');
0279                 obj.addHistory(mfir.getInfo('mfir', 'None'), [], [], []);
0280
0281             case 1
0282                 %%%%%%%%%%% one input %%%%%%%%%%%
0283                 %%%%%%%%%%% filename %%%%%%%%%%%
0284                 if ischar(varargin{1})
0285                     %%%%%%%%%%% f = mfir('foo.xml') %%%%%%%%%%%
0286                     %%%%%%%%%%% f = mfir('foo.mat') %%%%%%%%%%%
0287                     %----- filename
0288                     utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
0289                     obj = fromFile(obj, plist('filename', varargin{1}));
0290
0291                 elseif isa(varargin{1}, 'ao')
0292                     %%%%%%%%%%% f = mfir(ao-object) %%%%%%%%%%%
0293                     utils.helper.msg(msg.OPROC1, 'constructing from AO %s', varargin{1}.name);
0294                     %----- AO
0295                     pl = mfir.getDefaultPlist('From AO');
0296                     pl = pset(pl, 'AO', varargin{1});
0297                     obj = fromAO(obj, pl);
0298
0299                 elseif isstruct(varargin{1})
0300                     %%%%%%%%%%% f = mfir(struct) %%%%%%%%%%%
0301                     %----- struct
0302                     utils.helper.msg(msg.OPROC1, 'constructing from struct');
0303
0304                     %% Set properties which are declared in this class
0305                     obj.gd = varargin{1}.gd;
0306                     obj.version = varargin{1}.version;
0307
0308                 elseif isa(varargin{1}, 'plist')
0309                     %%%%%%%%%%% f = mfir(plist-object) %%%%%%%%%%%
0310                     %----- plist
0311                     pl = varargin{1};
0312                     filename = find(pl, 'filename');
0313                     hostname = find(pl, 'hostname');
0314                     conn = find(pl, 'conn');
0315                     type = find(pl, 'type');
0316                     pzm = find(pl, 'pzmmodel');
0317                     ipl = find(pl, 'iplist');

```



```

0322     a      = find(pl, 'a');
0323     aoin   = find(pl, 'AO');
0324
0325     % Selection of construction method
0326     if ~isempty(filename)
0327         %--- Construct from file
0328         utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
0329         obj = fromFile(obj, pl);
0330     elseif ~isempty(hostname) || ~isempty(conn)
0331         utils.helper.msg(msg.OPROC1, 'constructing from repository %s', hostname);
0332         %--- Construct from repository
0333         % do hostname constructor
0334         obj = obj.fromRepository(pl);
0335     elseif ~isempty(type)
0336         utils.helper.msg(msg.OPROC1, 'constructing standard %s filter', type);
0337         %--- Construct from standard type
0338         obj = fromStandard(obj, pl);
0339     elseif ~isempty(pzm)
0340         utils.helper.msg(msg.OPROC1, 'constructing from pzmodel %s', pzm.name);
0341         %--- Construct from pzmodel
0342         obj = fromPzmodel(obj, pl);
0343     elseif ~isempty(a)
0344         utils.helper.msg(msg.OPROC1, 'constructing from A/B coefficients');
0345         %--- Construct from a,b coeffs
0346         obj = fromA(obj, pl);
0347     elseif ~isempty(aoin)
0348         utils.helper.msg(msg.OPROC1, 'constructing from AO %s', aoin.name);
0349         %--- Construct from AO
0350         obj = fromAO(obj, pl);
0351     elseif ~isempty(ipl)
0352         %--- Construct from plist
0353         % if the plist is empty, we return an empty MFIR
0354         if nparams(ipl) == 0
0355             else
0356                 % do plist constructor
0357                 obj = mfir(ipl);
0358             end
0359         else
0360             %--- ERROR
0361             %%% is the plist is empty then return an empty MFIR
0362             if nparams(pl) == 0
0363                 else
0364                     error('### Unknown MFIR constructor method.');
```

```

0365             end
0366         end
0367     else
0368         error('### Unknown 1 argument constructor.');
```

```

0369     end
0370     case 2
0371         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0372         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% two input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0373         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0374
0375         if isa(varargin{1}, 'ao')
0376             %%%%%%%%% f = mfir(ao-object, plist-object) %%%%%%%%%
0377             utils.helper.msg(msg.OPROC1, 'constructing from AO %s', varargin{1}.name);
0378             obj = fromAO(obj, append(varargin{2}, 'AO', varargin{1}));
0379
0380         elseif isa(varargin{1}, 'database') && isnumeric(varargin{2})
0381             %%%%%%%%% f = mfir(<database-object>, [IDs]) %%%%%%%%%
0382             % mfir(<database-object>, [IDs])
0383             utils.helper.msg(msg.OPROC1, 'retrieve from repository');
0384             obj = obj.fromRepository(plist('conn', varargin{1}, 'id', varargin{2}));
0385
0386         elseif isa(varargin{1}, 'mfir') && isa(varargin{2}, 'plist') &&
0387             isempty(varargin{2}.params)
0388             % pass to copy constructor
0389             obj = mfir(varargin{1});
0390         elseif isnumeric(varargin{1})
0391             %%%%%%%%% f = mfir(a, fs) %%%%%%%%%
0392             %---- mfir(a, fs)
0393             utils.helper.msg(msg.OPROC1, 'constructing from A coefficients');
0394             obj = fromA(obj, plist('A', varargin{1}, 'fs', varargin{2}));
0395         else
0396             error('### Unknown 2 argument constructor.');
```

```

0397     end
0398     otherwise
0399         [firs, invars, rest] = utils.helper.collect_objects(args, 'mfir');
```

```

0400
0401         %%% Do we have a list of AOs as input
0402         if ~isempty(firs) && isempty(rest)
0403             obj = mfir(firs);
0404         else
0405             error('### Unknown number of arguments.');
```

```

0406     end
end
```

```

0407
0408     end % End constructor
0409 end % End public methods
0410
0411 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0412 %                                     Methods (static)                                     %
0413 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0414 methods (Static)
0415
0416     varargout = update_struct(varargin);
0417
0418     function out = VEROUT()
0419         out = ['$Id: mfir.m,v 1.63 2008/09/07 10:27:46 hewitson Exp $-->'
ltpda_filter.VEROUT];
0420     end
0421
0422     function ii = getInfo(varargin)
0423         ii = utils.helper.generic_getInfo(varargin{:}, 'mfir');
0424     end
0425
0426     function out = SETS()
0427         out = {...
0428             'Default',          ...
0429             'From XML File',    ...
0430             'From MAT File',    ...
0431             'From LISO File',   ...
0432             'From Repository',  ...
0433             'From Standard Type', ...
0434             'From Pzmodel',     ...
0435             'From A',           ...
0436             'From AO',          ...
0437             'From Plist'};
0438     end
0439
0440     function out = getDefaultPlist(set)
0441         switch set
0442         case 'Default'
0443             out = plist();
0444             %--- Repository constructor
0445         case 'From Repository'
0446             out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
0447             %--- Create from description
0448         case 'From Standard Type'
0449             order = 64;
0450             out = plist('type', 'lowpass', 'fc', 0.1, 'gain', 1.0, ...
0451                 'fs', 1.0, 'order', order, 'win', specwin('Hamming', order+1));
0452             %--- Create from a pzmodel
0453         case 'From Pzmodel'
0454             out = plist('pzmodel', pzmodel());
0455             %--- Create from frequency series AO
0456         case 'From AO'
0457             N = 512;
0458             out = plist('AO', ao(), 'N', N, 'method', 'frequency-sampling', ...
0459                 'win', specwin('Hanning', N+1));
0460             %--- Read from XML file
0461         case 'From XML File'
0462             out = plist('filename', '');
0463             %--- Read from MAT file
0464         case 'From MAT File'
0465             out = plist('filename', '');
0466             %--- Create from a plist
0467         case 'From Plist'
0468             out = plist('Plist', []);
0469             %--- From a coeffs
0470         case 'From A'
0471             out = plist('a', [], 'fs', []);
0472         otherwise
0473             out = plist();
0474         end
0475     end
0476
0477 end % End static methods
0478
0479 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0480 %                                     Methods (static, private)                               %
0481 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0482
0483 methods (Static, Access=private)
0484
0485     plo = parseFilterParams(pl)
0486
0487 end
0488
0489 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0490 %                                     Methods (public)                                     %
0491 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```



```
0492
0493 methods
0494     varargout = char(varargin);
0495     varargout = copy(varargin)
0496     varargout = display(varargin);
0497     varargout = string(varargin);
0498 end
0499
0500 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0501 %                                     Methods (protected)                                %
0502 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0503
0504 methods (Access = protected)
0505     obj = setGd(obj, val)
0506 end
0507
0508 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0509 %                                     Methods (private)                                %
0510 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0511
0512 methods (Access = private)
0513     f = fromPzmodel(f, pli)
0514     f = fromA(f, pli)
0515     f = fromAO(f, pli)
0516     f = fromStandard(f, pli)
0517
0518     f = mklowpass(f, pl)
0519     f = mkhighpass(f, pl)
0520     f = mkbandpass(f, pl)
0521     f = mkbandreject(f, pl)
0522 end
0523
0524 end
0525
```

redesign

PURPOSE ^

REDESIGN redesign the input filter to work for the given sample rate.

SYNOPSIS ^

function varargout = redesign(varargin)

DESCRIPTION ^

```
REDESIGN redesign the input filter to work for the given sample rate.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: REDESIGN redesign the input filter to work for the given sample
             rate.

CALL:       filt = redesign(filt, fs)

INPUT:      filt - input (mfir) filter
            fs   - new sample rate

OUTPUT:     filt - new output filter

M-FILE INFO: Get information about this methods by calling
              >> mfir.getInfo('redesign')

              Get information about a specified set-plist by calling:
              >> mfir.getInfo('redesign', 'set')

VERSION:    $Id: redesign.m,v 1.10 2008/09/04 15:29:30 ingo Exp $

HISTORY:    04-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [mfir](#) MFIR FIR filter object class constructor.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % REDESIGN redesign the input filter to work for the given sample rate.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: REDESIGN redesign the input filter to work for the given sample
0005 %             rate.
0006 %
0007 % CALL:       filt = redesign(filt, fs)
0008 %
0009 % INPUT:      filt - input (mfir) filter
0010 %            fs   - new sample rate
0011 %
```

```

0012 % OUTPUT:      filt - new output filter
0013 %
0014 % M-FILE INFO:  Get information about this methods by calling
0015 %               >> mfir.getInfo('redesign')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> mfir.getInfo('redesign', 'set')
0019 %
0020 % VERSION:      $Id: redesign.m,v 1.10 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:      04-04-2007 M Hewitson
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026
0027 function varargout = redesign(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 import utils.const.*
0036 utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0037
0038 %%% check number of outputs
0039 if nargout == 0
0040     error('### cat cannot be used as a modifier. Please give an output variable.');
```

```

0041 end
0042
0043 [filt, filt_invars, rest] = utils.helper.collect_objects(varargin(:), 'mfir');
0044
0045 if numel(rest) ~= 1 && ~isnumeric(rest{1})
0046     error('### Please specify the new sample rate.');
```

```

0047 end
0048 fs = rest{1};
0049
0050 %%% Redesing all filters
0051 for kk = 1:numel(filt)
0052     utils.helper.msg(msg.OPROC1, 're-designing filter for fs=%2.2f Hz...', fs);
0053
0054     fpl = filt(kk).hist.plistUsed;
0055     type = find(fpl, 'type');
0056     pzm = find(fpl, 'pzmodel');
```

```

0057
0058     if strcmpi(type, 'highpass') || ...
0059        strcmpi(type, 'lowpass') || ...
0060        strcmpi(type, 'bandpass') || ...
0061        strcmpi(type, 'bandreject')
0062
0063         utils.helper.msg(msg.OPROC2, 're-designing standard filter.');
```

```

0064     try
0065         fpl = pset(fpl, 'fs', fs);
0066         utils.helper.msg(msg.OPROC2, 'setting new fs.');
```

```

0067     catch
0068         fpl = append(fpl, 'fs', fs);
0069         utils.helper.msg(msg.OPROC2, 'appending parameter fs.');
```

```

0070     end
0071     filt(kk) = mfir(fpl);
0072     filt(kk).histout = filt(kk).histout;
0073 elseif ~isempty(pzm)
0074     utils.helper.msg(msg.OPROC2, 're-designing pzmodel filter.');
```

```

0075     filt(kk) = mfir(plist([param('pzmodel', pzm) param('fs', fs)]));
0076     filt(kk).histout = filt(kk).histout;
0077 else
0078     warning('!!! un-recognised input filter type. Can''t redesign.');
```

```

0079 end
0080 end
0081
0082 %%% Set output.
0083 varargout{1} = filt;
0084 end
0085
0086 %%%%%%%%%%%%%%
0087 %               Local Functions
0088 %%%%%%%%%%%%%%
0089
0090 %%%%%%%%%%%%%%
0091 %
0092 % FUNCTION:      getInfo
0093 %
0094 % DESCRIPTION:  Get Info Object
0095 %
0096 % HISTORY:      11-07-07 M Hewitson
0097 %               Creation.
```

```
0098 %
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100
0101 function ii = getInfo(varargin)
0102     if nargin == 1 && strcmpi(varargin{1}, 'None')
0103         sets = {};
0104         pl    = [];
0105     else
0106         sets = {'Default'};
0107         pl    = getDefaultPlist;
0108     end
0109     % Build info object
0110     ii = minfo(mfilename, 'mfir', '', utils.const.categories.helper, '$Id: redesign.m,v
1.10 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0111 end
0112
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114 %
0115 % FUNCTION:      getDefaultPlist
0116 %
0117 % DESCRIPTION:  Get Default Plist
0118 %
0119 % HISTORY:      11-07-07 M Hewitson
0120 %               Creation.
0121 %
0122 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0123
0124 function plo = getDefaultPlist()
0125     plo = plist();
0126 end
0127
```

setHistout

PURPOSE ^

SETHISTOUT Set the property 'histout'

SYNOPSIS ^

function varargout = setHistout(varargin)

DESCRIPTION ^

```
SETHISTOUT Set the property 'histout'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETHISTOUT Set the property 'histout'

CALL:      obj = obj.setHistout(2);
           obj = setHistout(obj, 2);

INPUTS:    obj - is a ltpda_filter object

M-FILE INFO: Get information about this methods by calling
              >> miir.getInfo('setHistout')

              Get information about a specified set-plist by calling:
              >> miir.getInfo('setHistout', 'set')

VERSION:    $Id: setHistout.m,v 1.3 2008/09/04 15:29:30 ingo Exp $

HISTORY:    16-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_filter](#) LTPDA_FILTER is the abstract base class for ltpda filter objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETHISTOUT Set the property 'histout'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETHISTOUT Set the property 'histout'
0005 %
0006 % CALL:      obj = obj.setHistout(2);
0007 %           obj = setHistout(obj, 2);
0008 %
0009 % INPUTS:    obj - is a ltpda_filter object
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 %              >> miir.getInfo('setHistout')
0013 %
0014 %           Get information about a specified set-plist by calling:
```

```

0015 %                >> miir.getInfo('setHistout', 'set')
0016 %
0017 % VERSION:      $Id: setHistout.m,v 1.3 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      16-07-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%
0023
0024 function varargout = setHistout(varargin)
0025
0026     %% Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     obj = varargin{1};
0033     val = varargin{2};
0034
0035     %% decide whether we modify the ltpda_filter-object, or create a new one.
0036     obj = copy(obj, nargout);
0037
0038     %% set 'histout'
0039     obj.histout = val;
0040
0041     %% set output
0042     varargout{1} = obj;
0043 end
0044
0045
0046 %%%%%%%%%%%%%%
0047 %                Local Functions
0048 %%%%%%%%%%%%%% %
0049
0050 %%%%%%%%%%%%%%
0051 %
0052 % FUNCTION:      getInfo
0053 %
0054 % DESCRIPTION:   Get Info Object
0055 %
0056 % HISTORY:      11-07-07 M Hewitson
0057 %                Creation.
0058 %
0059 %%%%%%%%%%%%%%
0060
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl = [];
0065     else
0066         sets = {'Default'};
0067         pl = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'ltpda_filter', '', utils.const.categories.internal, '$Id:
setHistout.m,v 1.3 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0071 end
0072
0073 %%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getDefaultPlist
0076 %
0077 % DESCRIPTION:   Get Default Plist
0078 %
0079 % HISTORY:      11-07-07 M Hewitson
0080 %                Creation.
0081 %
0082 %%%%%%%%%%%%%%
0083
0084 function plo = getDefaultPlist()
0085     plo = plist('histout', '');
0086 end
0087

```

copy

PURPOSE ^

COPY Make copy of mfir objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

COPY Make copy of mfir objects depending of the second input
DESCRIPTION: COPY Make copy of mfir objects depending of the second input
CALL: objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
>> mfir.getInfo('copy')
VERSION: \$Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp \$
HISTORY: 11-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 mfir MFIR FIR filter object class constructor.

This function is called by:

 mfir MFIR FIR filter object class constructor.
 setGd SETGD Set the property 'gd'

SOURCE CODE ^

```
0001 % COPY Make copy of mfir objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of mfir objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> mfir.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function varargout = copy(old, deepcopy)
0019 %
0020     if deepcopy
0021         t0 = time;
0022         prov = provenance;
0023         s = size(old);
```

```
0024     obj(s(1),s(2)) = mfir;  
0025     for kk=1:numel(old)  
0026         utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);  
0027         obj(kk).gd      = old(kk).gd;  
0028         obj(kk).fs      = old(kk).fs;  
0029         obj(kk).infile  = old(kk).infile;  
0030         obj(kk).a       = old(kk).a;  
0031         obj(kk).histout = old(kk).histout;  
0032         obj(kk).hist    = old(kk).hist;  
0033         obj(kk).name    = old(kk).name;  
0034         % reset on copy  
0035         obj(kk).created = t0;  
0036         obj(kk).creator = prov;  
0037         obj(kk).version = mfir.VEROUT;  
0038     end  
0039 else  
0040     obj = old;  
0041 end  
0042 varargout{1} = obj;  
0043 end  
0044  
0045
```


char

PURPOSE ^

CHAR convert a mfir object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a mfir object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a mfir object into a string.

CALL:      string = char(mfir)

M-FILE INFO: Get information about this methods by calling
              >> mfir.getInfo('char')

              Get information about a specified set-plist by calling:
              >> mfir.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.7 2008/09/04 15:29:30 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a mfir object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a mfir object into a string.
0005 %
0006 % CALL:      string = char(mfir)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> mfir.getInfo('char')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> mfir.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.7 2008/09/04 15:29:30 ingo Exp $
0015 %
0016 % HISTORY:    04-02-2007 M Hewitson
0017 %              Creation
```

```

0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'mfir');
0030
0031 pstr = '';
0032
0033 for ii = 1:numel(objs)
0034     obj = objs(ii);
0035
0036     pstr = [pstr sprintf('%s(fs=%.2f, ntaps=%.2f', obj.name, obj.fs, obj.ntaps)];
0037     if numel(obj.a) > 5
0038         pstr = [pstr ', a=' mat2str(obj.a(1:5),3)];
0039         pstr = [pstr(1:end-1) ' ...]], '];
0040     else
0041         pstr = [pstr ', a=' mat2str(obj.a,3) '), '];
0042     end
0043 end
0044
0045 pstr = strrep(pstr, '_', '\_');
0046 pstr = pstr(1:end-2);
0047
0048 varargout{1} = pstr;
0049 end
0050
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052 %                                     Local Functions                                     %
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056 %
0057 % FUNCTION:      getInfo
0058 %
0059 % DESCRIPTION:   Get Info Object
0060 %
0061 % HISTORY:       11-07-07 M Hewitson
0062 %                 Creation.
0063 %
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065
0066 function ii = getInfo(varargin)
0067     if nargin == 1 && strcmpi(varargin{1}, 'None')
0068         sets = {};
0069         pl = [];
0070     else
0071         sets = {'Default'};
0072         pl = getDefaultPlist;
0073     end
0074     % Build info object
0075     ii = minfo(mfilename, 'mfir', '', utils.const.categories.output, '$Id: char.m,v 1.7
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0076 end
0077
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079 %
0080 % FUNCTION:      getDefaultPlist
0081 %
0082 % DESCRIPTION:   Get Default Plist
0083 %
0084 % HISTORY:       11-07-07 M Hewitson
0085 %                 Creation.
0086 %
0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088
0089 function plo = getDefaultPlist()
0090     plo = plist();
0091 end
0092

```

display

PURPOSE ^

DISPLAY overloads display functionality for mfir objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for mfir objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for mfir objects.

CALL:      txt  = display(mfir)

INPUT:     mfir - IIR filter object

OUTPUT:    txt  - cell array with strings to display the mfir object

M-FILE INFO: Get information about this methods by calling
              >> mfir.getInfo('display')

              Get information about a specified set-plist by calling:
              >> mfir.getInfo('display', 'set')

VERSION:   $Id: display.m,v 1.12 2008/09/04 15:29:30 ingo Exp $

HISTORY:   11-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for mfir objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for mfir objects.
0005 %
0006 % CALL:      txt  = display(mfir)
0007 %
0008 % INPUT:     mfir - IIR filter object
0009 %
0010 % OUTPUT:    txt  - cell array with strings to display the mfir object
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> mfir.getInfo('display')
```

```
0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> mfir.getInfo('display', 'set')
0017 %
0018 % VERSION:      $Id: display.m,v 1.12 2008/09/04 15:29:30 ingo Exp $
0019 %
0020 % HISTORY:      11-01-2007 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027     %% Check if this is a call for parameters
0028     if utils.helper.isinfocall(varargin{:})
0029         varargout{1} = getInfo(varargin{3});
0030         return
0031     end
0032
0033     objs = utils.helper.collect_objects(varargin(:), 'mfir');
0034
0035     % get display text
0036     txt = utils.helper.objdisp(objs);
0037
0038     if nargin == 0
0039         for ii=1:length(txt)
0040             disp(txt{ii});
0041         end
0042     else
0043         varargout{1} = txt;
0044     end
0045 end
0046
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048 %                               Local Functions
0049 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0050
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052 %
0053 % FUNCTION:      getInfo
0054 %
0055 % DESCRIPTION:   Get Info Object
0056 %
0057 % HISTORY:      11-07-07 M Hewitson
0058 %                Creation.
0059 %
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061
0062 function ii = getInfo(varargin)
0063     if nargin == 1 && strcmpi(varargin{1}, 'None')
0064         sets = {};
0065         pl = [];
0066     else
0067         sets = {'Default'};
0068         pl = getDefaultPlist;
0069     end
0070     % Build info object
0071     ii = minfo(mfilename, 'mfir', '', utils.const.categories.output, '$Id: display.m,v
1.12 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0072 end
0073
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075 %
0076 % FUNCTION:      getDefaultPlist
0077 %
0078 % DESCRIPTION:   Get Default Plist
0079 %
0080 % HISTORY:      11-07-07 M Hewitson
0081 %                Creation.
0082 %
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084
0085 function plo = getDefaultPlist()
0086     plo = plist();
0087 end
0088
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input filter object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input filter object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: STRING writes a command string that can be used to recreate the
             input filter object.
CALL:       cmd = string(mfir)
INPUTS:     cmd - IIR filter object
OUTPUTS:    mfir - command string to create the IIR filter object
M-FILE INFO: Get information about this methods by calling
              >> mfir.getInfo('string')
              Get information about a specified set-plist by calling:
              >> mfir.getInfo('string', 'None')
VERSION:    $Id: string.m,v 1.8 2008/09/04 15:29:30 ingo Exp $
HISTORY:    29-03-2007 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input filter object.

This function is called by:

-  [mfir](#) MFIR FIR filter object class constructor.
-  [string](#) STRING writes a command string that can be used to recreate the input filter object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input filter object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %              input filter object.
0006 %
0007 % CALL:       cmd = string(mfir)
```

```
0008 %
0009 % INPUTS:      cmd - IIR filter object
0010 %
0011 % OUTPUTS:     mfir - command string to create the IIR filter object
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> mfir.getInfo('string')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> mfir.getInfo('string', 'None')
0018 %
0019 % VERSION:     $Id: string.m,v 1.8 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY:     29-03-2007 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%
0025 %
0026 function varargout = string(varargin)
0027 %
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 objs = utils.helper.collect_objects(varargin(:), 'mfir');
0035
0036 cmd = '';
0037 for kk = 1:numel(objs)
0038     infile = objs(kk).infile;
0039     if ~isempty(infile)
0040         cmd = [cmd 'mfir('' infile ''), '];
0041     else
0042         pl = objs(kk).hist.plistUsed;
0043         cmd = [cmd 'mfir(' string(pl) '), '];
0044     end
0045 end
0046
0047 cmd = cmd(1:end-2);
0048
0049 if numel(objs) > 1
0050     cmd = ['[' cmd ']'];
0051 end
0052
0053 varargout{1} = cmd;
0054 end
0055
0056 %%%%%%%%%%%%%%
0057 %                               Local Functions                               %
0058 %%%%%%%%%%%%%%
0059 %
0060 %%%%%%%%%%%%%%
0061 %
0062 % FUNCTION:      getInfo
0063 %
0064 % DESCRIPTION:   Get Info Object
0065 %
0066 % HISTORY:      11-07-07 M Hewitson
0067 %               Creation.
0068 %
0069 %%%%%%%%%%%%%%
0070
0071 function ii = getInfo(varargin)
0072     if nargin == 1 && strcmpi(varargin{1}, 'None')
0073         sets = {};
0074         pl = [];
0075     else
0076         sets = {'Default'};
0077         pl = getDefaultPlist;
0078     end
0079     % Build info object
0080     ii = minfo(mfilename, 'mfir', '', utils.const.categories.output, '$Id: string.m,v 1.8
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0081 end
0082
0083 %%%%%%%%%%%%%%
0084 %
0085 % FUNCTION:      getDefaultPlist
0086 %
0087 % DESCRIPTION:   Get Default Plist
0088 %
0089 % HISTORY:      11-07-07 M Hewitson
0090 %               Creation.
0091 %
0092 %%%%%%%%%%%%%%
```

```
0093  
0094 function plo = getDefaultPlist()  
0095     plo = plist();  
0096 end  
0097
```

resp

PURPOSE ^

RESP Make a frequency response of the filter.

SYNOPSIS ^

function [varargout] = resp(varargin)

DESCRIPTION ^

```
RESP Make a frequency response of the filter.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RESP Make a frequency response of the filter.
             The input filter should be a mfir object.

             The response is returned as a frequency-series in an analysis object.

             If no outputs are specified, the frequency-series is plotted as
             mag/deg.

CALL:
      a = resp(filt);
      a = resp(filt, f);
      a = resp(filt, pl);

PARAMETERS: 'f1'      - the start frequency
             'f2'      - the stop frequency
             'nf'       - number of evaluation points
             'scale'    - spacing of frequencies: 'lin' or 'log'
or           'f'       - a vector of frequency values

OUTPUTS:    a          - an analysis object

The response is calculated as:

      H(s) = gain * (a0 + a1*exp(s/fs) + ... + an*exp((n-1)s/fs))

M-FILE INFO: Get information about this methods by calling
              >> mfir.getInfo('resp')

              Get information about a specified set-plist by calling:
              >> mfir.getInfo('resp', 'List')

VERSION:     $Id: resp.m,v 1.20 2008/09/04 15:29:30 ingo Exp $

HISTORY:     27-08-02 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function out = getDefaultPlist\(set\)](#)

SOURCE CODE ^


```

0001 % RESP Make a frequency response of the filter.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RESP Make a frequency response of the filter.
0005 %               The input filter should be a mfir object.
0006 %
0007 %               The response is returned as a frequency-series in an analysis object.
0008 %
0009 %               If no outputs are specified, the frequency-series is plotted as
0010 %               mag/deg.
0011 %
0012 % CALL:         a = resp(filt);
0013 %               a = resp(filt, f);
0014 %               a = resp(filt, pl);
0015 %
0016 % PARAMETERS:  'f1'      - the start frequency
0017 %               'f2'      - the stop frequency
0018 %               'nf'      - number of evaluation points
0019 %               'scale'   - spacing of frequencies: 'lin' or 'log'
0020 %               or
0021 %               'f'       - a vector of frequency values
0022 %
0023 % OUTPUTS:     a         - an analysis object
0024 %
0025 % The response is calculated as:
0026 %
0027 %             
$$H(s) = \text{gain} * (a_0 + a_1 \exp(s/fs) + \dots + a_n \exp((n-1)s/fs))$$

0028 %
0029 % M-FILE INFO:  Get information about this methods by calling
0030 %               >> mfir.getInfo('resp')
0031 %
0032 %               Get information about a specified set-plist by calling:
0033 %               >> mfir.getInfo('resp', 'List')
0034 %
0035 % VERSION:     $Id: resp.m,v 1.20 2008/09/04 15:29:30 ingo Exp $
0036 %
0037 % HISTORY:     27-08-02 M Hewitson
0038 %               Creation
0039 %
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041
0042 function [varargout] = resp(varargin)
0043
0044 %%% Check if this is a call for parameters
0045 if utils.helper.isinfocall(varargin{:})
0046     varargout{1} = getInfo(varargin{3});
0047     return
0048 end
0049
0050 %%% get input filter
0051 filt = varargin{1};
0052 if ~isa(filt, 'mfir')
0053     error('### first input must be an mfir filter object.');
```

```

0086     end
0087     switch scale
0088     case 'lin'
0089         f = linspace(f1, f2, ndata);
0090     case 'log'
0091         f = logspace(log10(f1), log10(f2), ndata);
0092     end
0093 else
0094     % Compute from frequency list
0095     f = find(pl, 'f');
0096     % Want to deal with rows
0097     reshape_f = false;
0098     if size(f,1) > 1
0099         f = f.';
0100     end
0101     reshape_f = true;
0102     ndata = length(f);
0103 end
0104
0105 % create output parameter list, with the frequency array values and the
0106 % input filter
0107 plo = plist('f', f, 'filter', filt);
0108
0109 %%% compute Laplace vector
0110 s = -1i.*2*pi.*f;
0111
0112 %%% Compute filter response
0113 dresp = zeros(1, ndata);
0114
0115 for n=1:length(filt.a)
0116     dresp = dresp + filt.a(n).*exp(s.*(n-1)/filt.fs);
0117 end
0118
0119 %%% Create an analysis object
0120 if reshape_f
0121     f = f.';
0122     dresp = dresp.';
0123 end
0124
0125 % create new output fsdata
0126 fs = fsdata(f, dresp, filt.fs);
0127
0128 % make output analysis object
0129 b = ao(fs);
0130
0131 % Add history
0132 b.addHistory(getInfo, plo, [], filt.hist);
0133
0134 % set name
0135 b.setName(sprintf('resp(%s)', filt.name), 'internal');
0136
0137 % Outputs
0138 if nargout == 0
0139     ipplot(b)
0140 end
0141
0142 if nargout == 1
0143     varargout{1} = b;
0144 end
0145 if nargout > 1
0146     error('incorrect output arguments');
0147 end
0148 end
0149
0150 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0151 %                                     Local Functions                                     %
0152 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0153
0154 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0155 %
0156 % FUNCTION:      getInfo
0157 %
0158 % DESCRIPTION:  Get Info Object
0159 %
0160 % HISTORY:      11-07-07 M Hewitson
0161 %               Creation.
0162 %
0163 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0164
0165 function ii = getInfo(varargin)
0166     if nargin == 1 && strcmpi(varargin{1}, 'None')
0167         sets = {};
0168         pls = [];
0169     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0170         sets{1} = varargin{1};
0171         pls = getDefaultPlist(sets{1});

```

```

0172 else
0173     sets = {'List', 'Range'};
0174     pls = [];
0175     for kk=1:numel(sets)
0176         pls = [pls getDefaultPlist(sets{kk})];
0177     end
0178 end
0179 % Build info object
0180 ii = minfo(mfilename, 'mfir', '', utils.const.categories.sigproc, '$Id: resp.m,v 1.20
2008/09/04 15:29:30 ingo Exp $', sets, pls);
0181 end
0182
0183 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0184 %
0185 % FUNCTION:      getDefaultPlist
0186 %
0187 % DESCRIPTION:  Get Default Plist
0188 %
0189 % HISTORY:      11-07-07 M Hewitson
0190 %               Creation.
0191 %
0192 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0193
0194 function out = getDefaultPlist(set)
0195 switch set
0196 case 'List'
0197     out = plist('f', []);
0198 case 'Range'
0199     out = plist('f1', -1, 'f2', -1, 'nf', 1000, 'scale', 'log');
0200 otherwise
0201     error('### Unknown set [%s] for a default list.', set);
0202 end
0203 end
0204

```

miir

PURPOSE ^

MIIR IIR filter object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
MIIR IIR filter object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MIIR IIR filter object class constructor.
              Create a miir object.

SUPER CLASSES: ltpda_filter < ltpda_uoh < ltpda_uo < ltpda_obj

PROPERTIES:

  Inherit Properties (read only)
    name      - name of object
    created   - creation time (time-object)
    prov      - contains a instance of the provenance class.
    hist      - history of the object (history object)
    version   - cvs-version string.
    fs        - sample rate that the filter is designed for
    infile    - filename if the filter was loaded from file
    a         - set of numerator coefficients
    histout   - output history values to filter

  Protected Properties (read only)
    ntps      - number of coefficients in the filter
    b         -
    histin    - input history values to filter

MIIR Methods:

  Defined Abstract methods:
    char      - returns one character string which represents the object
    copy      - copies an object
    display   - displays an object
    string    - converts an object to a command string which will
                recreate the plist object.
    update_struct - updates a object structure to the current tbx-version

  Public methods:
    setHistin - set the property 'histin'
    redesign  - redesign the input filter to work for the given sample rate.
    resp      - make a frequency response of the filter.

  Protected methods:
    setB      - set the property 'b'

  Private methods:
    fromPzmodel - construct an miir from a pzmodel
    fromAB      - construct an miir from coefficients
    fromStandard - construct an miir from a standard types
    fromFile    - construct an miir filter from a file
    fromRepository - construct an miir filter from a repository
    mklowpass   - return a low pass filter
    mkhighpass  - return a high pass filter
    mkbandpass  - return a bandpass filter
    mkbandreject - return a low pass filter

CONSTRUCTORS:

    f = miir()          - creates an empty miir object.
    f = miir(fi)        - creates a copy of the input miir object, fi.
    f = miir(a,b,fs)    - creates a miir object from the coefficient
                        vectors 'a' and 'b' **.
                        The sample rate for which the filter is
```

```

    f = miir('foo_iir.fil') - create a miir object from a
                             LISO IIR .fil file.
    f = miir('foo_iir.xml') - create a miir object loading the miir object
                             from disk.
    f = miir('foo_iir.mat') - create a miir object loading the miir object
                             from disk.
    f = miir(pl)             - create a miir object from the description
                             given in the parameter list.

```

Parameter sets for plist constructor (in order of priority):

From XML File

Construct an MIIR by loading it from an XML file.

```

'filename' - construct an MIIR from a filename.
Example: plist('filename', 'm1.xml')
[default: empty string]

```

From MAT File

Construct an MIIR by loading it from a MAT file.

```

'filename' - construct an MIIR from a filename.
Example: plist('filename', 'm1.mat')
[default: empty string]

```

From LISO File

Construct an MIIR by loading it from a LISO file.

```

'filename' - construct an MIIR from a filename.
Example: plist('filename', 'm1.fil')
[default: empty string]

```

From Repository

Construct an MIIR by retrieving it from an LTPDA repository.

```

'Hostname' - the repository hostname. Only those objects which
              are MIIRs are returned.
[default: 'localhost'];

Additional parameters:
'Database' - The database name [default: 'ltpda']
'ID'       - A vector of object IDs. [default: []]
'CID'      - Retrieve all MIIR objects from a particular
              collection.

```

From Standard Type

Construct an MIIR of a standard type.

```

'type'      - one of the types: 'highpass', 'lowpass',
                          'bandpass', 'bandreject' [default: 'lowpass']

You can also specify optional parameters:
'gain'      - the gain of the filter [default: 1]
'fc'        - the roll-off frequency [default: 0.1 Hz]
'fs'        - the sampling frequency to design for [default: 1 Hz]
'order'     - the filter order [default: 1]
'ripple'    - pass/stop-band ripple for bandpass
              and bandreject filters [default: 0.5]

```

From Pzmodel

Construct an MIIR from a pzmodel.

```

'pzmodel'   - a pzmodel object to construct the filter from [default: empty pzmodel]

```

From Plist

```

'Plist'     - construct from a plist. The value passed should be a plist
              object.
[default: empty plist]

```

EXAMPLE 1: Create an order 1 highpass filter with high frequency gain 2.
Filter is designed for 10 Hz sampled data and has a cut-off

```
frequency of 0.2 Hz.

>> pl = plist('type', 'highpass', ...
              'order', 1, ...
              'gain', 2.0, ...
              'fs', 10, ...
              'fc', 0.2);

>> f = miir(pl)

NOTES:  ** The convention used here for naming the filter coefficients is
         the opposite to MATLAB's convention. The recursion formula
         for this convention is

         b(1)*y(n) = a(1)*x(n) + a(2)*x(n-1) + ... + a(na+1)*x(n-na)
                   - b(2)*y(n-1) - ... - b(nb+1)*y(n-nb)

M-FILE INFO: The following call returns an minfo object that contains
              information about the miir constructor:
              >> info = miir.getInfo
or            >> info = miir.getInfo('miir')

              You can get information about class methods by calling:
              >> info = miir.getInfo(method)
e.g. >> info = miir.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
              >> info = miir.getInfo(method, set)
e.g. >> info = miir.getInfo('miir', 'From Pzmodel')

VERSION:  $Id: miir.m,v 1.75 2008/09/07 10:27:46 hewitson Exp $

HISTORY:  09-02-2007 M Hewitson
           Creation
           11-02-2008 M Hueller
           Help fixed
           Default parameter list defined

SEE ALSO:  mfir, ltpda_filter, ltpda_uoh, ltpda_uo, ltpda_obj, plist

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a miir object into a string.
-  [copy](#) COPY Make copy of miir objects depending of the second input
-  [display](#) DISPLAY overloads display functionality for miir objects.
-  [filload](#) Load a LISO *_iir.fil file to get the filter taps and return a
-  [fromAB](#) Construct an miir from coefficients
-  [fromLISO](#) FROMLISO Construct a miir filter from a LISO file
-  [fromPzmodel](#) Construct an miir from a pzmodel
-  [fromStandard](#) Construct an miir from a standard types
-  [miir](#) MIIR IIR filter object class constructor.
-  [mkbandpass](#) MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
-  [mkbandreject](#) MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.
-  [mkhighpass](#) MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.
-  [mklowpass](#) MKLOWPASS return a low pass filter miir(). A Butterworth filter is used.
-  [parseFilterParams](#) PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard IIR filter.
-  [redesign](#) REDESIGN redesign the input filter to work for the given sample rate.
-  [resp](#) RESP Make a frequency response of the filter.
-  [setB](#) SETB Set the property 'b'
-  [setHistin](#) SETHISTIN Set the property 'histin'
-  [string](#) STRING writes a command string that can be used to recreate the input filter object.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  **copy** COPY Make copy of miir objects depending of the second input
-  **miir** MIIR IIR filter object class constructor.
-  **redesign** REDESIGN redesign the input filter to work for the given sample rate.

SUBFUNCTIONS ^

-  **function** obj = set.histin(obj, val)
-  **function** obj = set.b(obj, val)
-  **function** obj = set.ntaps(obj, val)
-  **function** val = get.ntaps(obj)
-  **function** obj = miir(varargin)
-  **function** out = VEROUT()
-  **function** ii = getInfo(varargin)
-  **function** out = SETS()
-  **function** out = getDefaultPlist(set)

SOURCE CODE ^

```
0001 % MIIR IIR filter object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MIIR IIR filter object class constructor.
0005 %             Create a miir object.
0006 %
0007 % SUPER CLASSES: ltpda_filter < ltpda_uoh < ltpda_uo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         name      - name of object
0013 %         created    - creation time (time-object)
0014 %         prov       - contains a instance of the provenance class.
0015 %         hist       - history of the object (history object)
0016 %         version    - cvs-version string.
0017 %         fs         - sample rate that the filter is designed for
0018 %         infile     - filename if the filter was loaded from file
0019 %         a          - set of numerator coefficients
0020 %         histout    - output history values to filter
0021 %
0022 %     Protected Properties (read only)
0023 %         ntaps      - number of coefficients in the filter
0024 %         b          -
0025 %         histin     - input history values to filter
0026 %
0027 % MIIR Methods:
0028 %
0029 %     Defined Abstract methods:
0030 %         char        - returns one character string which represents the object
0031 %         copy        - copies an object
0032 %         display     - displays an object
0033 %         string      - converts an object to a command string which will
0034 %                     recreate the plist object.
0035 %         update_struct - updates a object structure to the current tbx-version
0036 %
0037 %     Public methods:
0038 %         setHistin    - set the property 'histin'
0039 %         redesign     - redesign the input filter to work for the given sample rate.
0040 %         resp         - make a frequency response of the filter.
0041 %
0042 %     Protected methods:
0043 %         setB         - set the property 'b'
0044 %
0045 %     Private methods:
0046 %         fromPzmodel  - construct an miir from a pzmodel
0047 %         fromAB       - construct an miir from coefficients
0048 %         fromStandard - construct an miir from a standard types
0049 %         fromFile     - construct an miir filter from a file
0050 %         fromRepository - construct an miir filter from a repository
0051 %         mklowpass    - return a low pass filter
0052 %         mkhighpass   - return a high pass filter
0053 %         mkbandpass   - return a bandpass filter
```



```

0054 %      mkbandreject - return a low pass filter
0055 %
0056 % CONSTRUCTORS:
0057 %
0058 %      f = miir() - creates an empty miir object.
0059 %      f = miir(fi) - creates a copy of the input miir object, fi.
0060 %      f = miir(a,b,fs) - creates a miir object from the coefficient
0061 %                        vectors 'a' and 'b' **.
0062 %                        The sample rate for which the filter is
0063 %                        designed should be specified as well.
0064 %      f = miir('foo_iir.fil') - create a miir object from a
0065 %                               LISO IIR .fil file.
0066 %      f = miir('foo_iir.xml') - create a miir object loading the miir object
0067 %                               from disk.
0068 %      f = miir('foo_iir.mat') - create a miir object loading the miir object
0069 %                               from disk.
0070 %      f = miir(pl) - create a miir object from the description
0071 %                   given in the parameter list.
0072 %
0073 % Parameter sets for plist constructor (in order of priority):
0074 %
0075 % From XML File
0076 % -----
0077 %
0078 % Construct an MIIR by loading it from an XML file.
0079 %
0080 % 'filename' - construct an MIIR from a filename.
0081 % Example: plist('filename', 'm1.xml')
0082 % [default: empty string]
0083 %
0084 % From MAT File
0085 % -----
0086 %
0087 % Construct an MIIR by loading it from a MAT file.
0088 %
0089 % 'filename' - construct an MIIR from a filename.
0090 % Example: plist('filename', 'm1.mat')
0091 % [default: empty string]
0092 %
0093 % From LISO File
0094 % -----
0095 %
0096 % Construct an MIIR by loading it from a LISO file.
0097 %
0098 % 'filename' - construct an MIIR from a filename.
0099 % Example: plist('filename', 'm1.fil')
0100 % [default: empty string]
0101 %
0102 % From Repository
0103 % -----
0104 %
0105 % Construct an MIIR by retrieving it from an LTPDA repository.
0106 %
0107 % 'Hostname' - the repository hostname. Only those objects which
0108 %              are MIIRs are returned.
0109 % [default: 'localhost'];
0110 %
0111 % Additional parameters:
0112 % 'Database' - The database name [default: 'ltpda']
0113 % 'ID' - A vector of object IDs. [default: []]
0114 % 'CID' - Retrieve all MIIR objects from a particular
0115 %         collection.
0116 %
0117 % From Standard Type
0118 % -----
0119 %
0120 % Construct an MIIR of a standard type.
0121 %
0122 % 'type' - one of the types: 'highpass', 'lowpass',
0123 % 'bandpass', 'bandreject' [default: 'lowpass']
0124 %
0125 % You can also specify optional parameters:
0126 % 'gain' - the gain of the filter [default: 1]
0127 % 'fc' - the roll-off frequency [default: 0.1 Hz]
0128 % 'fs' - the sampling frequency to design for [default: 1 Hz]
0129 % 'order' - the filter order [default: 1]
0130 % 'ripple' - pass/stop-band ripple for bandpass
0131 %            and bandreject filters [default: 0.5]
0132 %
0133 % From Pzmodel
0134 % -----
0135 %
0136 % Construct an MIIR from a pzmodel.
0137 %
0138 % 'pzmodel' - a pzmodel object to construct the filter from [default: empty
pzmodel]

```



```

0139 %
0140 % From Plist
0141 % -----
0142 %
0143 % 'Plist' - construct from a plist. The value passed should be a plist
0144 % object.
0145 % [default: empty plist]
0146 %
0147 % EXAMPLE 1: Create an order 1 highpass filter with high frequency gain 2.
0148 % Filter is designed for 10 Hz sampled data and has a cut-off
0149 % frequency of 0.2 Hz.
0150 %
0151 % >> pl = plist('type', 'highpass', ...
0152 %               'order', 1, ...
0153 %               'gain', 2.0, ...
0154 %               'fs', 10, ...
0155 %               'fc', 0.2);
0156 % >> f = miir(pl)
0157 %
0158 % NOTES: ** The convention used here for naming the filter coefficients is
0159 % the opposite to MATLAB's convention. The recursion formula
0160 % for this convention is
0161 %
0162 % 
$$b(1)*y(n) = a(1)*x(n) + a(2)*x(n-1) + \dots + a(na+1)*x(n-na)$$

0163 % 
$$- b(2)*y(n-1) - \dots - b(nb+1)*y(n-nb)$$

0164 %
0165 % M-FILE INFO: The following call returns an minfo object that contains
0166 % information about the miir constructor:
0167 % >> info = miir.getInfo
0168 % or >> info = miir.getInfo('miir')
0169 %
0170 % You can get information about class methods by calling:
0171 % >> info = miir.getInfo(method)
0172 % e.g. >> info = miir.getInfo('eq')
0173 %
0174 % You can also restrict the sets of parameters contained in
0175 % the minfo object by calling:
0176 % >> info = miir.getInfo(method, set)
0177 % e.g. >> info = miir.getInfo('miir', 'From Pzmodel')
0178 %
0179 % VERSION: $Id: miir.m,v 1.75 2008/09/07 10:27:46 hewitson Exp $
0180 %
0181 % HISTORY: 09-02-2007 M Hewitson
0182 %           Creation
0183 %           11-02-2008 M Hueller
0184 %           Help fixed
0185 %           Default parameter list defined
0186 %
0187 % SEE ALSO: mfir, ltpda_filter, ltpda_uoh, ltpda_uo, ltpda_obj, plist
0188 %
0189 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0190 %
0191 classdef miir < ltpda_filter
0192 %
0193 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0194 % Property definition %
0195 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0196 %----- Public (read/write) Properties -----
0197 properties
0198 end
0199
0200 %----- Protected read-only Properties -----
0201 properties (SetAccess = protected)
0202 b = [];
0203 histin = [];
0204 version = '$Id: miir.m,v 1.75 2008/09/07 10:27:46 hewitson Exp $';
0205 end
0206
0207 %----- Protected read-only Properties -----
0208 properties (SetAccess = protected, Dependent = true)
0209 ntps
0210 end
0211
0212 %----- Private Properties -----
0213 properties (GetAccess = protected, SetAccess = protected)
0214 end
0215
0216 %----- Abstract Properties -----
0217 properties (Abstract = true, SetAccess = protected)
0218 end
0219
0220 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0221 % Check property setting %
0222 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0223 %
0224 %

```

```

0225 methods
0226 function obj = set.histin(obj, val)
0227 if ~isempty(val)
0228     if ~isnumeric(val) || ~isreal(val)
0229         error('### The value for the property 'histin' must be a real number(s)');
0230     end
0231 end
0232 obj.histin = val;
0233 end
0234 function obj = set.b(obj, val)
0235 if ~isempty(val)
0236     if ~isnumeric(val) || ~isreal(val)
0237         error('### The value for the property 'b' must be a real number(s)');
0238     end
0239 end
0240 obj.b = val;
0241 end
0242 function obj = set.ntaps(obj, val)
0243 error('### Don't set the property 'ntaps'. It is computed by max(length(a),
length(b)).');
0244 end
0245 end
0246
0247 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0248 % Compute the Dependent properties %
0249 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0250
0251 methods
0252 function val = get.ntaps(obj)
0253 val = max(length(obj.a), length(obj.b));
0254 end
0255 end
0256
0257 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0258 % Constructor %
0259 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0260
0261 methods
0262 function obj = miir(varargin)
0263
0264     import utils.const.*
0265     utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0266
0267     %%% Call superclass
0268     obj = obj@ltpda_filter(varargin{:});
0269
0270     %%% Set default values %%%
0271
0272     % Collect all miir objects
0273     [fs, invars, rest] = utils.helper.collect_objects(varargin{:}, 'miir');
0274
0275     if isempty(rest) && ~isempty(fs)
0276         % Do copy constructor and return
0277         utils.helper.msg(msg.OPROC1, 'copy constructor');
0278         obj = copy(fs, 1);
0279         for kk=1:numel(obj)
0280             obj(kk).addHistory(miir.getInfo('miir', 'None'), [], [], obj(kk).hist);
0281         end
0282         return
0283     end
0284
0285     if nargin == 0
0286         utils.helper.msg(msg.OPROC1, 'empty constructor');
0287         obj.addHistory(miir.getInfo('miir', 'None'), [], [], []);
0288     elseif nargin == 1
0289         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0290         %%% one input %%%
0291         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0292
0293         if ischar(varargin{1})
0294
0295             %%% f = miir('foo.mat') %%%
0296             %%% f = miir('foo.xml') %%%
0297             %----- filename
0298             utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
0299             obj = fromFile(obj, plist('filename', varargin{1}));
0300
0301         elseif isstruct(varargin{1})
0302             %%% f = miir(struct) %%%
0303             %----- struct
0304             utils.helper.msg(msg.OPROC1, 'constructing from struct');
0305
0306             %%% Set properties which are declared in this class
0307             obj.b = varargin{1}.b;
0308             obj.histin = varargin{1}.histin;
0309             obj.version = varargin{1}.version;

```

```

0310
0311 elseif isa(varargin{1}, 'plist')
0312     %%%%%%%%% f = miir(plist-object) %%%%%%%%%
0313     %----- plist
0314     pl = varargin{1};
0315     filename = find(pl, 'filename');
0316     hostname = find(pl, 'hostname');
0317     conn = find(pl, 'conn');
0318     type = find(pl, 'type');
0319     pzm = find(pl, 'pzmodel');
0320     ipl = find(pl, 'plist');
0321     a = find(pl, 'a');
0322
0323     % Selection of construction method
0324     if ~isempty(filename)
0325         utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
0326         %--- Construct from file
0327         obj = fromFile(obj, pl);
0328     elseif ~isempty(hostname) || ~isempty(conn)
0329         utils.helper.msg(msg.OPROC1, 'constructing from repository %s', hostname);
0330         %--- Construct from repository
0331         % do hostname constructor
0332         obj = obj.fromRepository(pl);
0333     elseif ~isempty(type)
0334         utils.helper.msg(msg.OPROC1, 'constructing from standard %s', type);
0335         %--- Construct from standard type
0336         obj = fromStandard(obj, pl);
0337     elseif ~isempty(pzm)
0338         utils.helper.msg(msg.OPROC1, 'constructing from pzmodel %s', pzm.name);
0339         %--- Construct from pzmodel
0340         obj = fromPzmodel(obj, pl);
0341     elseif ~isempty(a)
0342         utils.helper.msg(msg.OPROC1, 'constructing from A/B coefficients');
0343         %--- Construct from a,b coeffs
0344         obj = fromAB(obj, pl);
0345     elseif ~isempty(ipl)
0346         %--- Construct from plist
0347         % if the plist is empty, we return an empty MIIR
0348         if nparams(ipl) == 0
0349             else
0350                 % do plist constructor
0351                 obj = miir(ipl);
0352             end
0353         else
0354             %--- ERROR
0355             %%% is the plist is empty then return an empty MIIR
0356             if nparams(pl) == 0
0357                 else
0358                     error('### Unknown MIIR constructor method.');
```

```

0393
0394 elseif nargin == 3
0395     %%%%%%%%%%% three input %%%%%%%%%%%
0396     %%%%%%%%%%% three input %%%%%%%%%%%
0397     %%%%%%%%%%% three input %%%%%%%%%%%
0398
0399     utils.helper.msg(msg.OPROC1, 'constructing from A/B coefficients');
0400     %%%%%%%%%%% f = miir(a, b, fs) %%%%%%%%%%%
0401     % a,b,fs constructor
0402     obj = fromAB(obj, plist('a', varargin{1}, 'b', varargin{2}, 'fs', varargin{3}));
0403
0404 else
0405     [iirs, invars, rest] = utils.helper.collect_objects(args, 'miir');
0406
0407     %%% Do we have a list of AOs as input
0408     if ~isempty(iirs) && isempty(rest)
0409         obj = miir(iirs);
0410     else
0411         error('### Unknown number of constructor arguments.');

```

```
0479
0480 end % End static methods
0481
0482 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0483 %                               Methods (static, private) %
0484 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0485
0486 methods (Static, Access=private)
0487
0488     f = filload(filename)
0489     plo = parseFilterParams(pl)
0490
0491 end % End static, private methods
0492
0493 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0494 %                               Methods (public) %
0495 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0496
0497 methods
0498     varargout = char(varargin);
0499     varargout = copy(varargin)
0500     varargout = display(varargin);
0501     varargout = string(varargin);
0502
0503     varargout = setHistin(varargin)
0504     varargout = redesign(varargin)
0505     varargout = resp(varargin)
0506 end
0507
0508 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0509 %                               Methods (protected) %
0510 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0511
0512 methods (Access = protected)
0513     obj = setB(obj, val)
0514     obj = fromLISO(obj, pli)
0515 end
0516
0517 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0518 %                               Methods (private) %
0519 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0520
0521 methods (Access = private)
0522     f = fromPzmodel(f, pli)
0523     f = fromAB(f, pli)
0524     f = fromStandard(f, pli)
0525
0526     f = mklowpass(f, pl)
0527     f = mkhighpass(f, pl)
0528     f = mkbandpass(f, pl)
0529     f = mkbandreject(f, pl)
0530 end
0531
0532 end % End classdef
0533
0534 % END
0535
```

redesign

PURPOSE ^

REDESIGN redesign the input filter to work for the given sample rate.

SYNOPSIS ^

function varargout = redesign(varargin)

DESCRIPTION ^

```
REDESIGN redesign the input filter to work for the given sample rate.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: REDESIGN redesign the input filter to work for the given sample
             rate.

CALL:       filt = redesign(filt, fs)

INPUT:      filt - input (miir) filter
            fs   - new sample rate

OUTPUT:     filt - new output filter

M-FILE INFO: Get information about this methods by calling
              >> miir.getInfo('redesign')

              Get information about a specified set-plist by calling:
              >> miir.getInfo('redesign', 'set')

VERSION:    $Id: redesign.m,v 1.15 2008/09/04 15:29:31 ingo Exp $

HISTORY:    04-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [miir](#) MIIR IIR filter object class constructor.

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % REDESIGN redesign the input filter to work for the given sample rate.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: REDESIGN redesign the input filter to work for the given sample
0005 %             rate.
0006 %
0007 % CALL:       filt = redesign(filt, fs)
0008 %
```



```

0009 % INPUT:      filt - input (miir) filter
0010 %            fs   - new sample rate
0011 %
0012 % OUTPUT:     filt - new output filter
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> miir.getInfo('redesign')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %               >> miir.getInfo('redesign', 'set')
0019 %
0020 % VERSION:    $Id: redesign.m,v 1.15 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:    04-04-2007 M Hewitson
0023 %              Creation
0024 %
0025 %%%%%%%%%%%%%%%
0026
0027 function varargout = redesign(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 import utils.const.*
0036 utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0037
0038 %%% check number of outputs
0039 if nargin == 0
0040     error('### cat cannot be used as a modifier. Please give an output variable.');
```

```

0041 end
0042
0043 %%% Get miir objects
0044 [filt, filt_invars, rest] = utils.helper.collect_objects(varargin(:), 'miir');
0045
0046 if numel(rest) ~= 1 && ~isnumeric(rest{1})
0047     error('### Please specify the new sample rate.');
```

```

0048 end
0049 fs = rest{1};
0050
0051 %%% Redesing all filters
0052 for kk = 1:numel(filt)
0053     utils.helper.msg(msg.OPROC1, 're-designing filter for fs=%2.2f Hz...', fs);
0054     fpl = filt(kk).hist.plistUsed;
0055     type = find(fpl, 'type');
0056     pzm = find(fpl, 'pzmodel');
```

```

0057
0058     if strcmpi(type, 'highpass') || ...
0059        strcmpi(type, 'lowpass') || ...
0060        strcmpi(type, 'bandpass') || ...
0061        strcmpi(type, 'bandreject')
0062
0063         utils.helper.msg(msg.OPROC2, 're-designing standard filter.');
```

```

0064         try
0065             fpl = pset(fpl, 'fs', fs);
0066             utils.helper.msg(msg.OPROC2, 'setting new fs.');
```

```

0067         catch
0068             fpl = append(fpl, 'fs', fs);
0069             utils.helper.msg(msg.OPROC2, 'appending parameter fs.');
```

```

0070         end
0071         filt(kk) = miir(fpl);
0072         filt(kk).setHistin(filt(kk).histin);
0073         filt(kk).setHistout(filt(kk).histout);
0074     elseif ~isempty(pzm)
0075
0076         utils.helper.msg(msg.OPROC2, 're-designing pzmodel filter.');
```

```

0077         filt(kk) = miir(plist('pzmodel', pzm, 'fs', fs));
0078         filt(kk).setHistin(filt(kk).histin);
0079         filt(kk).setHistout(filt(kk).histout);
0080     else
0081         warning('!!! un-recognised input filter type. Can't redesign.');
```

```

0082     end
0083 end
0084
0085 %%% Set output.
0086 varargout{1} = filt;
0087
0088 end
0089
0090 %%%%%%%%%%%%%%%
0091 %                               Local Functions                               %
0092 %%%%%%%%%%%%%%%
0093
0094 %%%%%%%%%%%%%%%

```

```
0095 %
0096 % FUNCTION:      getInfo
0097 %
0098 % DESCRIPTION: Get Info Object
0099 %
0100 % HISTORY:        11-07-07 M Hewitson
0101 %                  Creation.
0102 %
0103 %%%%%%%%%%%%%%%
0104
0105 function ii = getInfo(varargin)
0106     if nargin == 1 && strcmpi(varargin{1}, 'None')
0107         sets = {};
0108         pl    = [];
0109     else
0110         sets = {'Default'};
0111         pl    = getDefaultPlist;
0112     end
0113     % Build info object
0114     ii = minfo(mfilename, 'miir', '', utils.const.categories.helper, '$Id: redesign.m,v
1.15 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0115 end
0116
0117 %%%%%%%%%%%%%%%
0118 %
0119 % FUNCTION:      getDefaultPlist
0120 %
0121 % DESCRIPTION: Get Default Plist
0122 %
0123 % HISTORY:        11-07-07 M Hewitson
0124 %                  Creation.
0125 %
0126 %%%%%%%%%%%%%%%
0127
0128 function plo = getDefaultPlist()
0129     plo = plist();
0130 end
0131
```


copy

PURPOSE ^

COPY Make copy of miir objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

COPY Make copy of miir objects depending of the second input
DESCRIPTION: COPY Make copy of miir objects depending of the second input
CALL: objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
>> miir.getInfo('copy')
VERSION: \$Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp \$
HISTORY: 11-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 miir MIIR IIR filter object class constructor.

This function is called by:

-  miir MIIR IIR filter object class constructor.
-  setB SETB Set the property 'b'
-  setHistin SETHISTIN Set the property 'histin'

SOURCE CODE ^

```
0001 % COPY Make copy of miir objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of miir objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> miir.getInfo('copy')
0010 %
0011 % VERSION:   $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:   11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function varargout = copy(old, deepcopy)
0019
0020
0021     if deepcopy
0022         t0 = time;
```

```
0023     prov = provenance;
0024     s     = size(old);
0025     obj(s(1),s(2)) = miir;
0026     for kk=1:numel(old)
0027         utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
0028         obj(kk).fs      = old(kk).fs;
0029         obj(kk).infile  = old(kk).infile;
0030         obj(kk).a       = old(kk).a;
0031         obj(kk).b       = old(kk).b;
0032         obj(kk).histin  = old(kk).histin;
0033         obj(kk).histout = old(kk).histout;
0034         obj(kk).hist    = old(kk).hist;
0035         obj(kk).name    = old(kk).name;
0036         obj(kk).created = t0;
0037         obj(kk).creator = prov;
0038         obj(kk).version = miir.VEROUT;
0039     end
0040 else
0041     obj = old;
0042 end
0043 varargout{1} = obj;
0044 end
0045
```

setHistin

PURPOSE ^

SETHISTIN Set the property 'histin'

SYNOPSIS ^

function varargout = setHistin(varargin)

DESCRIPTION ^

```
SETHISTIN Set the property 'histin'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETHISTIN Set the property 'histin'

CALL:      obj = obj.setHistin(2);
           obj = setHistin(obj, 2);

INPUTS:    obj - is a miir object

M-FILE INFO: Get information abin this methods by calling
              >> miir.getInfo('setHistin')

              Get information abin a specified set-plist by calling:
              >> miir.getInfo('setHistin', 'set')

VERSION:    $Id: setHistin.m,v 1.3 2008/09/04 15:29:31 ingo Exp $

HISTORY:    16-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of miir objects depending of the second input

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETHISTIN Set the property 'histin'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETHISTIN Set the property 'histin'
0005 %
0006 % CALL:      obj = obj.setHistin(2);
0007 %           obj = setHistin(obj, 2);
0008 %
0009 % INPUTS:    obj - is a miir object
0010 %
0011 % M-FILE INFO: Get information abin this methods by calling
```

```
0012 %           >> miir.getInfo('setHistin')
0013 %
0014 %           Get information abin a specified set-plist by calling:
0015 %           >> miir.getInfo('setHistin', 'set')
0016 %
0017 % VERSION:      $Id: setHistin.m,v 1.3 2008/09/04 15:29:31 ingo Exp $
0018 %
0019 % HISTORY:      16-07-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setHistin(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the miir-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'histin'
0039 obj.histin = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                               Local Functions                               %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'ltpda_filter', '', utils.const.categories.internal, '$Id:
setHistin.m,v 1.3 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('histin', '');
0084 end
0085
```

char

PURPOSE ^

CHAR convert a miir object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a miir object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a miir object into a string.

CALL:      string = char(miir)

M-FILE INFO: Get information about this methods by calling
              >> miir.getInfo('char')

              Get information about a specified set-plist by calling:
              >> miir.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.10 2008/09/04 15:29:31 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a miir object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a miir object into a string.
0005 %
0006 % CALL:      string = char(miir)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> miir.getInfo('char')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> miir.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.10 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:    04-02-2007 M Hewitson
0017 %              Creation
```

```

0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'miir');
0030
0031 pstr = '';
0032
0033 for ii = 1:numel(objs)
0034     obj = objs(ii);
0035
0036     pstr = [pstr sprintf('%s(fs=%.2f, ntaps=%.2f', obj.name, obj.fs, obj.ntaps)];
0037     pstr = [pstr ', a=' mat2str(obj.a,3)];
0038     pstr = [pstr ', b=' mat2str(obj.b,3) '), '];
0039 end
0040
0041 pstr = strrep(pstr, '-', '\_');
0042 pstr = pstr(1:end-2);
0043
0044 varargout{1} = pstr;
0045 end
0046
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048 % Local Functions %
0049 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0050
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052 %
0053 % FUNCTION:      getInfo
0054 %
0055 % DESCRIPTION:   Get Info Object
0056 %
0057 % HISTORY:       11-07-07 M Hewitson
0058 %                Creation.
0059 %
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061
0062 function ii = getInfo(varargin)
0063     if nargin == 1 && strcmpi(varargin{1}, 'None')
0064         sets = {};
0065         pl = [];
0066     else
0067         sets = {'Default'};
0068         pl = getDefaultPlist;
0069     end
0070     % Build info object
0071     ii = minfo(mfilename, 'miir', '', utils.const.categories.output, '$Id: char.m,v 1.10
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0072 end
0073
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075 %
0076 % FUNCTION:      getDefaultPlist
0077 %
0078 % DESCRIPTION:   Get Default Plist
0079 %
0080 % HISTORY:       11-07-07 M Hewitson
0081 %                Creation.
0082 %
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084
0085 function plo = getDefaultPlist()
0086     plo = plist();
0087 end
0088

```

display

PURPOSE ^

DISPLAY overloads display functionality for miir objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for miir objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for miir objects.

CALL:      txt  = display(miir)

INPUT:     miir - IIR filter object

OUTPUT:    txt  - cell array with strings to display the miir object

M-FILE INFO: Get information about this methods by calling
              >> miir.getInfo('display')

              Get information about a specified set-plist by calling:
              >> miir.getInfo('display', 'set')

VERSION:    $Id: display.m,v 1.16 2008/09/04 15:29:31 ingo Exp $

HISTORY:    11-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for miir objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for miir objects.
0005 %
0006 % CALL:      txt  = display(miir)
0007 %
0008 % INPUT:     miir - IIR filter object
0009 %
0010 % OUTPUT:    txt  - cell array with strings to display the miir object
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> miir.getInfo('display')
```

```
0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> miir.getInfo('display', 'set')
0017 %
0018 % VERSION:      $Id: display.m,v 1.16 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY:      11-01-2007 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027     %% Check if this is a call for parameters
0028     if utils.helper.isinfocall(varargin{:})
0029         varargout{1} = getInfo(varargin{3});
0030         return
0031     end
0032
0033     %% Get miir objects
0034     objs = utils.helper.collect_objects(varargin(:), 'miir');
0035
0036     %% get display text
0037     txt = utils.helper.objdisp(objs);
0038
0039     if nargin == 0
0040         for ii=1:length(txt)
0041             disp(txt{ii});
0042         end
0043     else
0044         varargout{1} = txt;
0045     end
0046 end
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %                                Local Functions                                %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053 %
0054 % FUNCTION:      getInfo
0055 %
0056 % DESCRIPTION:   Get Info Object
0057 %
0058 % HISTORY:      11-07-07 M Hewitson
0059 %                Creation.
0060 %
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062
0063 function ii = getInfo(varargin)
0064     if nargin == 1 && strcmpi(varargin{1}, 'None')
0065         sets = {};
0066         pl   = [];
0067     else
0068         sets = {'Default'};
0069         pl   = getDefaultPlist;
0070     end
0071     % Build info object
0072     ii = minfo(mfilename, 'miir', '', utils.const.categories.output, '$Id: display.m,v
1.16 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0073 end
0074
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076 %
0077 % FUNCTION:      getDefaultPlist
0078 %
0079 % DESCRIPTION:   Get Default Plist
0080 %
0081 % HISTORY:      11-07-07 M Hewitson
0082 %                Creation.
0083 %
0084 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0085
0086 function plo = getDefaultPlist()
0087     plo = plist();
0088 end
0089
```


string

PURPOSE ^

STRING writes a command string that can be used to recreate the input filter object.

SYNOPSIS ^

```
function varargout = string(varargin)
```

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input filter object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate the
             input filter object.

CALL:       cmd = string(miir)

INPUTS:     cmd - IIR filter object

OUTPUTS:    miir - command string to create the IIR filter object

M-FILE INFO: Get information about this methods by calling
              >> miir.getInfo('string')

              Get information about a specified set-plist by calling:
              >> miir.getInfo('string', 'None')

VERSION:    $Id: string.m,v 1.9 2008/09/04 15:29:31 ingo Exp $

HISTORY:    29-03-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input filter object.

This function is called by:

-  [miir](#) MIIR IIR filter object class constructor.
-  [string](#) STRING writes a command string that can be used to recreate the input filter object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input filter object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input filter object.
0006 %
0007 % CALL:       cmd = string(miir)
```

```
0008 %
0009 % INPUTS:      cmd - IIR filter object
0010 %
0011 % OUTPUTS:     miir - command string to create the IIR filter object
0012 %
0013 % M-FILE INFO:  Get information about this methods by calling
0014 %               >> miir.getInfo('string')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> miir.getInfo('string', 'None')
0018 %
0019 % VERSION:      $Id: string.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
0020 %
0021 % HISTORY:      29-03-2007 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%
0025 %
0026 function varargout = string(varargin)
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 %%% Get miir objects
0035 objs = utils.helper.collect_objects(varargin(:), 'miir');
0036
0037 cmd = '';
0038 for kk = 1:numel(objs)
0039     infile = objs(kk).infile;
0040     if ~isempty(infile)
0041         cmd = [cmd 'miir('' infile ''), '];
0042     else
0043         pl = objs(kk).hist.plistUsed;
0044         cmd = [cmd 'miir(' string(pl) '), '];
0045     end
0046 end
0047
0048 cmd = cmd(1:end-2);
0049
0050 if numel(objs) > 1
0051     cmd = ['[' cmd ']'];
0052 end
0053
0054 varargout{1} = cmd;
0055 end
0056
0057 %%%%%%%%%%%%%%
0058 %               Local Functions
0059 %%%%%%%%%%%%%%
0060 %%%%%%%%%%%%%%
0061 %
0062 % FUNCTION:      getInfo
0063 %
0064 % DESCRIPTION:   Get Info Object
0065 %
0066 % HISTORY:       11-07-07 M Hewitson
0067 %               Creation.
0068 %
0069 %
0070 %%%%%%%%%%%%%%
0071 %
0072 function ii = getInfo(varargin)
0073     if nargin == 1 && strcmpi(varargin{1}, 'None')
0074         sets = {};
0075         pl = [];
0076     else
0077         sets = {'Default'};
0078         pl = getDefaultPlist;
0079     end
0080     % Build info object
0081     ii = minfo(mfilename, 'miir', '', utils.const.categories.output, '$Id: string.m,v 1.9
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0082 end
0083
0084 %%%%%%%%%%%%%%
0085 %
0086 % FUNCTION:      getDefaultPlist
0087 %
0088 % DESCRIPTION:   Get Default Plist
0089 %
0090 % HISTORY:       11-07-07 M Hewitson
0091 %               Creation.
0092 %
```

```
0093 %%%%%%%%%%  
0094  
0095 function plo = getDefaultPlist()  
0096     plo = plist();  
0097 end  
0098
```

resp

PURPOSE ^

RESP Make a frequency response of the filter.

SYNOPSIS ^

function varargout = resp(varargin)

DESCRIPTION ^

```
RESP Make a frequency response of the filter.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RESP Make a frequency response of the filter.
             The input filter should be an miir object.

             The response is returned as a frequency-series in an
             analysis object.

             If no outputs are specified, the frequency-series is plotted as
             mag/deg.

CALL:       a = resp(filt, pl)
             resp(filt, pl);

PARAMETERS: 'f1'      - the start frequency
             'f2'      - the stop frequency
             'nf'       - number of evaluation points
             'scale'    - spacing of frequencies: 'lin' or 'log'
or          'f'        - a vector of frequency values

OUTPUTS:    a        - an analysis object

The response is calculated as:

             a0 + a1*exp(s/fs) + ... + an*exp((n-1)s/fs)
H(s) = gain * -----
             b0 + b1*exp(s/fs) + ... + bm*exp((m-1)s/fs)

M-FILE INFO: Get information about this methods by calling
             >> miir.getInfo('resp')

             Get information about a specified set-plist by calling:
             >> miir.getInfo('resp', 'List')

VERSION:    $Id: resp.m,v 1.20 2008/09/04 15:29:31 ingo Exp $

NOTE:       Some of the code below is taken from Mathworks's treeplot.m

HISTORY:    27-08-02 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SUBFUNCTIONS ^

 function ii = getInfo(varargin)
 function plo = getDefaultPlist(set)

SOURCE CODE ^

```

0001 % RESP Make a frequency response of the filter.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RESP Make a frequency response of the filter.
0005 %               The input filter should be an miir object.
0006 %
0007 %               The response is returned as a frequency-series in an
0008 %               analysis object.
0009 %
0010 %               If no outputs are specified, the frequency-series is plotted as
0011 %               mag/deg.
0012 %
0013 % CALL:         a = resp(filt, pl)
0014 %               resp(filt, pl);
0015 %
0016 % PARAMETERS: 'f1'      - the start frequency
0017 %              'f2'      - the stop frequency
0018 %              'nf'       - number of evaluation points
0019 %              'scale'    - spacing of frequencies: 'lin' or 'log'
0020 %              or
0021 %              'f'        - a vector of frequency values
0022 %
0023 % OUTPUTS:      a        - an analysis object
0024 %
0025 % The response is calculated as:
0026 %
0027 %
0028 %               
$$H(s) = \text{gain} * \frac{a_0 + a_1 \exp(s/fs) + \dots + a_n \exp((n-1)s/fs)}{b_0 + b_1 \exp(s/fs) + \dots + b_m \exp((m-1)s/fs)}$$

0029 %
0030 %
0031 %
0032 % M-FILE INFO:  Get information about this methods by calling
0033 %               >> miir.getInfo('resp')
0034 %
0035 %               Get information about a specified set-plist by calling:
0036 %               >> miir.getInfo('resp', 'List')
0037 %
0038 % VERSION:      $Id: resp.m,v 1.20 2008/09/04 15:29:31 ingo Exp $
0039 %
0040 % NOTE:         Some of the code below is taken from Mathworks's treeplot.m
0041 %
0042 % HISTORY:      27-08-02 M Hewitson
0043 %               Creation
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 function varargout = resp(varargin)
0048
0049 %%% Check if this is a call for parameters
0050 if utils.helper.isinfocall(varargin{:})
0051     varargout{1} = getInfo(varargin{3});
0052     return
0053 end
0054
0055 %%% get input filter
0056 filt = varargin{1};
0057 if ~isa(filt, 'miir')
0058     error('### first input must be an miir filter object.');
```

```

0078 if isempty(find(pl, 'f'))
0079 % Compute from frequency range
0080 f1 = find(pl, 'f1');
0081 f2 = find(pl, 'f2');
0082 ndata = find(pl, 'nf');
0083 scale = find(pl, 'scale');
0084
0085 % Set frequencies
0086 if f1 < 0
0087     f1 = filt.fs/1000;
0088 end
0089 if f2 < 0
0090     f2 = filt.fs/2-1/ndata;
0091 end
0092 switch scale
0093     case 'lin'
0094         f = linspace(f1, f2, ndata);
0095     case 'log'
0096         f = logspace(log10(f1), log10(f2), ndata);
0097 end
0098 else
0099 % Compute from frequency list
0100     f = find(pl, 'f');
0101 % Want to deal with rows
0102 reshape_f = false;
0103 if size(f,1) > 1
0104     f = f.';
0105     reshape_f = true;
0106 end
0107 ndata = length(f);
0108 end
0109
0110 % create output parameter list, with the frequency array values and the
0111 % input filter
0112 plo = plist('f', f, 'filter', filt);
0113
0114 %%% compute Laplace vector
0115 s = -1i.*2*pi.*f;
0116
0117 %%% Compute filter response
0118 num = zeros(1, ndata);
0119 for n=1:length(filt.a)
0120     num = num + filt.a(n).*exp(s.*(n-1)/filt.fs);
0121 end
0122 denom = zeros(1, ndata);
0123 for n=1:length(filt.b)
0124     denom = denom + filt.b(n).*exp(s.*(n-1)/filt.fs);
0125 end
0126 dresp = num ./ denom;
0127
0128 % mag = 20*log10(abs(dresp));
0129 % phase = angle(dresp)*180/pi;
0130
0131 %%% Create an analysis object
0132 if reshape_f
0133     f = f.';
0134     dresp = dresp.';
0135 end
0136
0137 % create new output fsdata
0138 fs = fsdata(f, dresp, filt.fs);
0139
0140 % make output analysis object
0141 b = ao(fs);
0142
0143 % Add history
0144 b.addHistory(getInfo, plo, [], filt.hist);
0145
0146 % set name
0147 b.setName(sprintf('resp(%s)', filt.name), 'internal');
0148
0149 % Outputs
0150 if nargout == 0
0151     ipplot(b)
0152 end
0153
0154 if nargout == 1
0155     varargout{1} = b;
0156 end
0157 if nargout > 1
0158     error('incorrect output arguments');
0159 end
0160 end
0161
0162 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0163 % Local Functions %

```

```
0164 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0165 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0166 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0167 %
0168 % FUNCTION:      getInfo
0169 %
0170 % DESCRIPTION:   Get Info Object
0171 %
0172 % HISTORY:       11-07-07 M Hewitson
0173 %                Creation.
0174 %
0175 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0176 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0177 function ii = getInfo(varargin)
0178     if nargin == 1 && strcmpi(varargin{1}, 'None')
0179         sets = {};
0180         pls  = [];
0181     elseif nargin == 1&& ~isempty(varargin{1}) && ischar(varargin{1})
0182         sets{1} = varargin{1};
0183         pls = getDefaultPlist(sets{1});
0184     else
0185         sets = {'List', 'Range'};
0186         pls = [];
0187         for kk=1:numel(sets)
0188             pls = [pls getDefaultPlist(sets{kk})];
0189         end
0190     end
0191     % Build info object
0192     ii = minfo(mfilename, 'miir', '', utils.const.categories.sigproc, '$Id: resp.m,v 1.20
2008/09/04 15:29:31 ingo Exp $', sets, pls);
0193 end
0194 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0195 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0196 %
0197 % FUNCTION:      getDefaultPlist
0198 %
0199 % DESCRIPTION:   Get Default Plist
0200 %
0201 % HISTORY:       11-07-07 M Hewitson
0202 %                Creation.
0203 %
0204 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0205 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0206 function plo = getDefaultPlist(set)
0207     switch set
0208     case 'List'
0209         plo = plist('f', []);
0210     case 'Range'
0211         plo = plist('f1', -1, 'f2', -1, 'nf', 1000, 'scale', 'log');
0212     otherwise
0213         error('### Unknown set [%s] for a default list.', set);
0214     end
0215 end
0216 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input pzmodel object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input pzmodel object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate
             the input pzmodel object.

CALL:       cmd = string(pzmodel)

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('string')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('string', 'None')

VERSION:    $Id: string.m,v 1.10 2008/09/04 15:29:31 ingo Exp $

HISTORY:    02-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input pzmodel object.

This function is called by:

-  [pzmodel](#) PZMODEL constructor for pzmodel class.
-  [string](#) STRING writes a command string that can be used to recreate the input pzmodel object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input pzmodel object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate
0005 %              the input pzmodel object.
0006 %
0007 % CALL:       cmd = string(pzmodel)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> pzmodel.getInfo('string')
0011 %
```



```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> pzmodel.getInfo('string', 'None')
0014 %
0015 % VERSION:      $Id: string.m,v 1.10 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      02-04-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = string(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 objs = utils.helper.collect_objects(varargin(:), 'pzmodel');
0031
0032 cmd = '';
0033 for kk = 1:numel(objs)
0034
0035     if ~isempty(objs(kk).hist)
0036         cmd = [cmd, sprintf('pzmodel(%s), ', string(objs(kk).hist.plistUsed))];
0037     else
0038         pl = plist();
0039         fields = fieldnames(objs(kk));
0040         for ii = 1:length(fields)
0041             field = fields{ii};
0042             pl = append(pl, field, objs(kk).(field));
0043         end
0044
0045         cmd = [cmd 'pzmodel(' string(pl) '), '];
0046     end
0047
0048 end
0049
0050 cmd = cmd(1:end-2);
0051
0052 if numel(objs) > 1
0053     cmd = ['[' cmd ']'];
0054 end
0055
0056 varargout{1} = cmd;
0057
0058 end
0059
0060 %%%%%%%%%%%%%%
0061 %                               Local Functions                               %
0062 %%%%%%%%%%%%%%
0063
0064 %%%%%%%%%%%%%%
0065 %
0066 % FUNCTION:      getInfo
0067 %
0068 % DESCRIPTION:   Get Info Object
0069 %
0070 % HISTORY:      11-07-07 M Hewitson
0071 %                Creation.
0072 %
0073 %%%%%%%%%%%%%%
0074
0075 function ii = getInfo(varargin)
0076     if nargin == 1 && strcmpi(varargin{1}, 'None')
0077         sets = {};
0078         pl = [];
0079     else
0080         sets = {'Default'};
0081         pl = getDefaultPlist;
0082     end
0083     % Build info object
0084     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.helper, '$Id: string.m,v
1.10 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0085 end
0086
0087 %%%%%%%%%%%%%%
0088 %
0089 % FUNCTION:      getDefaultPlist
0090 %
0091 % DESCRIPTION:   Get Default Plist
0092 %
0093 % HISTORY:      11-07-07 M Hewitson
0094 %                Creation.
0095 %
0096 %%%%%%%%%%%%%%

```

```
0097  
0098 function plo = getDefaultPlist()  
0099     plo = plist();  
0100 end  
0101
```

copy

PURPOSE ^

COPY Make copy of pzmodel objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of pzmodel objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: COPY Make copy of pzmodel objects depending of the second input
CALL:      objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('copy')
VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
HISTORY:    11-07-2008 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of pzmodel objects depending of the second input
-  [pzmodel](#) PZMODEL constructor for pzmodel class.

This function is called by:

-  [copy](#) COPY Make copy of pzmodel objects depending of the second input
-  [pzmodel](#) PZMODEL constructor for pzmodel class.
-  [setGain](#) SETGAIN Set the property 'gain'
-  [setPoles](#) SETPOLES Set the property 'poles'
-  [setZeros](#) SETZEROS Set the property 'zeros'

SOURCE CODE ^

```
0001 % COPY Make copy of pzmodel objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of pzmodel objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %               >> pzmodel.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
```

```
0018 function varargout = copy(old, deepcopy)
0019
0020 if deepcopy
0021     t0 = time;
0022     prov = provenance;
0023     s = size(old);
0024     obj(s(1),s(2)) = pzmodel;
0025     for kk=1:numel(old)
0026         utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
0027         obj(kk).gain = old(kk).gain;
0028         if ~isempty(old(kk).poles)
0029             obj(kk).poles = copy(old(kk).poles,1);
0030         else
0031             obj(kk).poles = [];
0032         end
0033         if ~isempty(old(kk).zeros)
0034             obj(kk).zeros = copy(old(kk).zeros,1);
0035         else
0036             obj(kk).zeros = [];
0037         end
0038         obj(kk).hist = old(kk).hist;
0039         obj(kk).name = old(kk).name;
0040         obj(kk).created = t0;
0041         obj(kk).creator = prov;
0042         obj(kk).version = pzmodel.VEROUT;
0043     end
0044 else
0045     obj = old;
0046 end
0047 varargout{1} = obj;
0048 end
0049
0050
```

getlowerFreq

PURPOSE ^

GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.

SYNOPSIS ^

function varargout = getlowerFreq(varargin)

DESCRIPTION ^

```
GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: GETLOWERFREQ gets the frequency of the lowest pole or zero in the
             model.

CALL:       f = getlowerFreq(pzm);

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('getlowerFreq')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('getlowerFreq', 'None')

VERSION:    $Id: getlowerFreq.m,v 1.7 2008/09/04 15:29:31 ingo Exp $

HISTORY:    04-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [pzmodel](#) PZMODEL constructor for pzmodel class.
-  [resp](#) RESP returns the complex response of a pzmodel as an Analysis Object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: GETLOWERFREQ gets the frequency of the lowest pole or zero in the
0005 %             model.
0006 %
0007 % CALL:       f = getlowerFreq(pzm);
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %             >> pzmodel.getInfo('getlowerFreq')
0011 %
0012 %             Get information about a specified set-plist by calling:
0013 %             >> pzmodel.getInfo('getlowerFreq', 'None')
0014 %
```

```
0015 % VERSION:      $Id: getlowerFreq.m,v 1.7 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:       04-04-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = getlowerFreq(varargin)
0023
0024 %% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 pzm = varargin{1};
0031 poles = pzm.poles;
0032 zeros = pzm.zeros;
0033 np = length(poles);
0034 nz = length(zeros);
0035 f = 1e10;
0036 for j=1:np
0037     pole = poles(j);
0038     fc = pole.f;
0039     if fc < f
0040         f = fc;
0041     end
0042 end
0043
0044 for j=1:nz
0045     zero = zeros(j);
0046     fc = zero.f;
0047     if fc < f
0048         f = fc;
0049     end
0050 end
0051
0052 % Set outputs
0053 varargout{1} = f;
0054 end
0055
0056 %%%%%%%%%%%%%%
0057 % Local Functions %
0058 %%%%%%%%%%%%%%
0059
0060 %%%%%%%%%%%%%%
0061 %
0062 % FUNCTION:      getInfo
0063 %
0064 % DESCRIPTION:   Get Info Object
0065 %
0066 % HISTORY:       11-07-07 M Hewitson
0067 %                Creation.
0068 %
0069 %%%%%%%%%%%%%%
0070
0071 function ii = getInfo(varargin)
0072     if nargin == 1 && strcmpi(varargin{1}, 'None')
0073         sets = {};
0074         pl = [];
0075     else
0076         sets = {'Default'};
0077         pl = getDefaultPlist;
0078     end
0079     % Build info object
0080     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.internal, '$Id:
getlowerFreq.m,v 1.7 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0081 end
0082
0083 %%%%%%%%%%%%%%
0084 %
0085 % FUNCTION:      getDefaultPlist
0086 %
0087 % DESCRIPTION:   Get Default Plist
0088 %
0089 % HISTORY:       11-07-07 M Hewitson
0090 %                Creation.
0091 %
0092 %%%%%%%%%%%%%%
0093
0094 function plo = getDefaultPlist()
0095     plo = plist();
0096 end
0097
```


getupperFreq

PURPOSE ^

GETUPPERFREQ gets the frequency of the highest pole or zero in the model.

SYNOPSIS ^

function varargout = getupperFreq(varargin)

DESCRIPTION ^

```
GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: GETUPPERFREQ gets the frequency of the highest pole or zero in
the model.

CALL:      f = getupperFreq(pzm);

M-FILE INFO: Get information about this methods by calling
>> pzmodel.getInfo('getupperFreq')

            Get information about a specified set-plist by calling:
            >> pzmodel.getInfo('getupperFreq', 'None')

VERSION:    $Id: getupperFreq.m,v 1.7 2008/09/04 15:29:31 ingo Exp $

HISTORY:    04-04-2007 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input pzmodel.
-  [pzmodel](#) PZMODEL constructor for pzmodel class.
-  [resp](#) RESP returns the complex response of a pzmodel as an Analysis Object.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: GETUPPERFREQ gets the frequency of the highest pole or zero in
0005 %               the model.
0006 %
0007 % CALL:      f = getupperFreq(pzm);
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %               >> pzmodel.getInfo('getupperFreq')
0011 %
0012 %               Get information about a specified set-plist by calling:
```



```

0013 %                >> pzmodel.getInfo('getupperFreq', 'None')
0014 %
0015 % VERSION:      $Id: getupperFreq.m,v 1.7 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      04-04-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = getupperFreq(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 pzm = varargin{1};
0031 poles = pzm.poles;
0032 zeros = pzm.zeros;
0033 np = length(poles);
0034 nz = length(zeros);
0035 f = 0;
0036 for j=1:np
0037     pole = poles(j);
0038     fc = pole.f;
0039     if fc > f
0040         f = fc;
0041     end
0042 end
0043
0044 for j=1:nz
0045     zero = zeros(j);
0046     fc = zero.f;
0047     if fc > f
0048         f = fc;
0049     end
0050 end
0051
0052 % Set outputs
0053 varargout{1} = f;
0054 end
0055
0056 %%%%%%%%%%%%%%
0057 %                Local Functions
0058 %%%%%%%%%%%%%%
0059
0060 %%%%%%%%%%%%%%
0061 %
0062 % FUNCTION:      getInfo
0063 %
0064 % DESCRIPTION:   Get Info Object
0065 %
0066 % HISTORY:      11-07-07 M Hewitson
0067 %                Creation.
0068 %
0069 %%%%%%%%%%%%%%
0070
0071 function ii = getInfo(varargin)
0072     if nargin == 1 && strcmpi(varargin{1}, 'None')
0073         sets = {};
0074         pl = [];
0075     else
0076         sets = {'Default'};
0077         pl = getDefaultPlist;
0078     end
0079     % Build info object
0080     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.internal, '$Id:
getupperFreq.m,v 1.7 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0081 end
0082
0083 %%%%%%%%%%%%%%
0084 %
0085 % FUNCTION:      getDefaultPlist
0086 %
0087 % DESCRIPTION:   Get Default Plist
0088 %
0089 % HISTORY:      11-07-07 M Hewitson
0090 %                Creation.
0091 %
0092 %%%%%%%%%%%%%%
0093
0094 function plo = getDefaultPlist()
0095     plo = plist();
0096 end
0097

```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

pzm2ab

PURPOSE ^

PZM2AB convert pzmodel to IIR filter coefficients using bilinear transform.

SYNOPSIS ^

function varargout = pzm2ab(varargin)

DESCRIPTION ^

```
PZM2AB convert pzmodel to IIR filter coefficients using bilinear transform.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PZM2AB convert pzmodel to IIR filter coefficients using bilinear
transform.

CALL:      [a,b] = pzm2ab(pzm, fs)

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('pzm2ab')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('pzm2ab', 'None')

VERSION:    $Id: pzm2ab.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:    03-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [pzmodel](#) PZMODEL constructor for pzmodel class.
-  [tomiir](#) TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

SUBFUNCTIONS ^

-  [function \[a,b\] = cpolezero\(pf, pq, zf, zq, fs\)](#)
-  [function g = iirdcgain\(a,b\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % PZM2AB convert pzmodel to IIR filter coefficients using bilinear transform.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PZM2AB convert pzmodel to IIR filter coefficients using bilinear
0005 %               transform.
0006 %
0007 % CALL:      [a,b] = pzm2ab(pzm, fs)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %               >> pzmodel.getInfo('pzm2ab')
0011 %
```

```

0012 %           Get information about a specified set-plist by calling:
0013 %           >> pzmmodel.getInfo('pzm2ab', 'None')
0014 %
0015 % VERSION:      $Id: pzm2ab.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      03-04-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = pzm2ab(varargin)
0023
0024     import utils.const.*
0025
0026     %% Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     %% Collect input variable names
0033     in_names = cell(size(varargin));
0034     for ii = 1:nargin,in_names{ii} = inputname(ii);end
0035
0036     %% Collect all AOs and plists
0037     [pzm, pzm_invars, fs] = utils.helper.collect_objects(varargin(:), 'pzmmodel',
in_names);
0038
0039     %% Check inputs
0040     if numel(pzm) ~= 1
0041         error('### Please use only one PZ-Model.');
```

```

0097     utils.helper.msg(msg.OPROC2, 'computing complex pole');
0098     [ai,bi] = cp2iir(cpoles(j), fs);
0099     if ~isempty(ao)>0
0100         [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
0101     else
0102         ao = ai;
0103         bo = bi;
0104     end
0105 end
0106 else
0107     % do remaining czeros
0108     for j=length(cpoles)+1:length(czeros)
0109         utils.helper.msg(msg.OPROC2, 'computing complex zero');
0110         [ai,bi] = cz2iir(czeros(j), fs);
0111         if ~isempty(ao)>0
0112             [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
0113         else
0114             ao = ai;
0115             bo = bi;
0116         end
0117     end
0118 end
0119
0120 % Now do the real poles and zeros
0121 for j=1:np
0122     pole = poles(j);
0123     if isnan(pole.q) || pole.q < 0.5
0124         utils.helper.msg(msg.OPROC2, 'computing real pole');
0125         [ai,bi] = rp2iir(pole, fs);
0126         if ~isempty(ao)>0
0127             [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
0128         else
0129             ao = ai;
0130             bo = bi;
0131         end
0132     end
0133 end
0134
0135 for j=1:nz
0136     zero = zeros(j);
0137     if isnan(zero.q) || zero.q < 0.5
0138         utils.helper.msg(msg.OPROC2, 'computing real zero');
0139         [ai,bi] = rz2iir(zero, fs);
0140         if ~isempty(ao)>0
0141             [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
0142         else
0143             ao = ai;
0144             bo = bi;
0145         end
0146     end
0147 end
0148
0149 ao = ao.*gain;
0150
0151 varargout{1} = ao;
0152 varargout{2} = bo;
0153 end
0154
0155 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0156 %                                     Local Functions                                     %
0157 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0158
0159 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0160 %
0161 % FUNCTION:      cpolezero
0162 %
0163 % DESCRIPTION:   Return IIR filter coefficients for a complex pole and
0164 %                complex zero designed using the bilinear transform.
0165 %
0166 % CALL:          [a,b] = cpolezero(pf, pq, zf, zq, fs)
0167 %
0168 % HISTORY:       18-02-2007 M Hewitson
0169 %                Creation.
0170 %
0171 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0172
0173 function [a,b] = cpolezero(pf, pq, zf, zq, fs)
0174
0175     utils.helper.msg(msg.OPROC1, 'computing complex pole/zero pair');
0176
0177     wp = pf*2*pi;
0178     wp2 = wp^2;
0179     wz = zf*2*pi;
0180     wz2 = wz^2;
0181
0182     k = 4*fs*fs + 2*wp*fs/pq + wp2;

```

```
0183
0184     a(1) = (4*fs*fs + 2*wz*fs/zq + wz2)/k;
0185     a(2) = (2*wz2 - 8*fs*fs)/k;
0186     a(3) = (4*fs*fs - 2*wz*fs/zq + wz2)/k;
0187     b(1) = 1;
0188     b(2) = (2*wp2 - 8*fs*fs)/k;
0189     b(3) = (wp2 + 4*fs*fs - 2*wp*fs/pq)/k;
0190
0191     % normalise dc gain to 1
0192     g = iirdcgain(a,b);
0193     a = a / g;
0194 end
0195
0196
0197 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0198 %
0199 % FUNCTION:      cpolezero
0200 %
0201 % DESCRIPTION:   Work out the DC gain of an IIR filter given the coefficients.
0202 %
0203 % CALL:          g = iirdcgain(a,b)
0204 %
0205 % INPUTS:        a - numerator coefficients
0206 %                b - denominator coefficients
0207 %
0208 % OUTPUTS        g - gain
0209 %
0210 % HISTORY:       03-07-2002 M Hewitson
0211 %                Creation.
0212 %
0213 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0214
0215 function g = iirdcgain(a,b)
0216     suma = sum(a);
0217     if(length(b)>1)
0218         sumb = sum(b);
0219         g = suma / sumb;
0220     else
0221         g = suma;
0222     end
0223 end
0224
0225 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0226 %
0227 % FUNCTION:      getInfo
0228 %
0229 % DESCRIPTION:   Get Info Object
0230 %
0231 % HISTORY:       11-07-07 M Hewitson
0232 %                Creation.
0233 %
0234 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0235
0236 function ii = getInfo(varargin)
0237     if nargin == 1 && strcmpi(varargin{1}, 'None')
0238         sets = {};
0239         pl = [];
0240     else
0241         sets = {'Default'};
0242         pl = getDefaultPlist;
0243     end
0244     % Build info object
0245     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.internal, '$Id: pzm2ab.m,v
1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0246 end
0247
0248 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0249 %
0250 % FUNCTION:      getDefaultPlist
0251 %
0252 % DESCRIPTION:   Get Default Plist
0253 %
0254 % HISTORY:       11-07-07 M Hewitson
0255 %                Creation.
0256 %
0257 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0258
0259 function plo = getDefaultPlist()
0260     plo = plist();
0261 end
0262
```

tomfir

PURPOSE ^

TOMFIR approximates a pole/zero model with an FIR filter.

SYNOPSIS ^

function varargout = tomfir(varargin)

DESCRIPTION ^

TOMFIR approximates a pole/zero model with an FIR filter.

DESCRIPTION: TOMFIR approximates a pole/zero model with an FIR filter. The response of the pzmodel is computed using pzmodel/resp with the additional input parameter of param('f1', 0). The final frequency in the response is set automatically from the pzmodel/resp function if not specified as an input. This upper frequency is then taken as the Nyquist frequency and the sample rate of the corresponding fsdata A0 is set accordingly. The function then calls mfir() with the new fsdata A0 as input. The result is an FIR filter designed to have a magnitude response equal to the magnitude response of the pole/zero model. The filter has linear phase and the phase of the pzmodel is ignored.

CALL: f = tomfir(pzm)
f = tomfir(pzm, plist)

PARAMETERS: f2 - the upper frequency. Sets the Nyquist rate used in the conversion. [default taken from pzmodel/resp]
nf - Sets the number of frequency points used in the conversion [default: 1000]

M-FILE INFO: Get information about this methods by calling
>> pzmodel.getInfo('tomfir')

Get information about a specified set-plist by calling:
>> pzmodel.getInfo('tomfir', 'None')

VERSION: \$Id: tomfir.m,v 1.7 2008/09/04 15:29:31 ingo Exp \$

HISTORY: 16-08-07 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 [resp](#) RESP returns the complex response of a pzmodel as an Analysis Object.

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % TOMFIR approximates a pole/zero model with an FIR filter.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TOMFIR approximates a pole/zero model with an FIR filter.
0005 % The response of the pzmodel is computed using pzmodel/resp with
0006 % the additional input parameter of param('f1', 0). The final
0007 % frequency in the response is set automatically from the
0008 % pzmodel/resp function if not specified as an input. This upper
0009 % frequency is then taken as the Nyquist frequency and the
0010 % sample rate of the corresponding fsdata A0 is set accordingly.
0011 % The function then calls mfir() with the new fsdata A0 as input.
0012 % The result is an FIR filter designed to have a magnitude response
0013 % equal to the magnitude response of the pole/zero model. The filter
0014 % has linear phase and the phase of the pzmodel is ignored.
0015 %
0016 % CALL:      f = tomfir(pzm)
0017 %            f = tomfir(pzm, plist)
0018 %
0019 % PARAMETERS: f2    - the upper frequency. Sets the Nyquist rate used in the
0020 %                  conversion. [default taken from pzmodel/resp]
0021 %                  nf    - Sets the number of frequency points used in the conversion
0022 %                        [default: 1000]
0023 %
0024 % M-FILE INFO: Get information about this methods by calling
0025 %              >> pzmodel.getInfo('tomfir')
0026 %
0027 %              Get information about a specified set-plist by calling:
0028 %              >> pzmodel.getInfo('tomfir', 'None')
0029 %
0030 % VERSION:     $Id: tomfir.m,v 1.7 2008/09/04 15:29:31 ingo Exp $
0031 %
0032 % HISTORY:     16-08-07 M Hewitson
0033 %              Creation
0034 %
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 function varargout = tomfir(varargin)
0038
0039 %%% Check if this is a call for parameters
0040 if utils.helper.isinfocall(varargin{:})
0041     varargout{1} = getInfo(varargin{3});
0042     return
0043 end
0044
0045 pl = plist;
0046
0047 if nargin == 1
0048     pzm = varargin{1};
0049 elseif nargin == 2
0050     pzm = varargin{1};
0051     pl = varargin{2};
0052 else
0053     error('### Incorrect inputs.');
```

```
0054 end
0055
0056 % check design parameters
0057 fs = find(pl, 'fs');
0058
0059 r = resp(pzm, plist('f1', 0, 'f2', fs/2, 'nf', 1000));
0060
0061 % Set fs
0062 r.setFs(fs);
0063
0064 % compute filter
0065 f = mfir(r);
0066
0067 if nargout == 1
0068     varargout{1} = f;
0069 else
0070     error('### Incorrect outputs.');
```

```
0071 end
0072
0073 end
0074
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076 %                               Local Functions                               %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080 %
0081 % FUNCTION:      getInfo
0082 %
0083 % DESCRIPTION:  Get Info Object
0084 %
```



```
0085 % HISTORY:      11-07-07 M Hewitson
0086 %
0087 %
0088 %%%%%%%%%%
0089
0090 function ii = getInfo(varargin)
0091     if nargin == 1 && strcmpi(varargin{1}, 'None')
0092         sets = {};
0093         pl   = [];
0094     else
0095         sets = {'Default'};
0096         pl   = getDefaultPlist;
0097     end
0098     % Build info object
0099     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.op, '$Id: tomfir.m,v 1.7
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0100 end
0101
0102 %%%%%%%%%%
0103 %
0104 % FUNCTION:      getDefaultPlist
0105 %
0106 % DESCRIPTION:   Get Default Plist
0107 %
0108 % HISTORY:      11-07-07 M Hewitson
0109 %
0110 %
0111 %%%%%%%%%%
0112
0113 function plo = getDefaultPlist()
0114     plo = plist();
0115 end
0116
```

tomiir

PURPOSE ^

TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.

SYNOPSIS ^

function f = tomiir(varargin)

DESCRIPTION ^

```
TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: TOMIIR converts a pzmodel to an IIR filter using a bilinear
transform.

CALL:      f = tomiir(pzm, fs); % construct for this sample frequency fs
           f = tomiir(pzm, pl); % construct from plist

PARAMETERS: 'fs' - sample frequency

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('tomiir')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('tomiir', 'None')

VERSION:    $Id: tomiir.m,v 1.8 2008/09/04 15:29:31 ingo Exp $

HISTORY:    03-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [pzm2ab](#) PZM2AB convert pzmodel to IIR filter coefficients using bilinear transform.

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TOMIIR converts a pzmodel to an IIR filter using a bilinear
0005 % transform.
0006 %
0007 % CALL:      f = tomiir(pzm, fs); % construct for this sample frequency fs
0008 %           f = tomiir(pzm, pl); % construct from plist
0009 %
0010 % PARAMETERS: 'fs' - sample frequency
```

```

0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 % >> pzmodel.getInfo('tomiir')
0014 %
0015 % Get information about a specified set-plist by calling:
0016 % >> pzmodel.getInfo('tomiir', 'None')
0017 %
0018 % VERSION: $Id: tomiir.m,v 1.8 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY: 03-04-2007 M Hewitson
0021 % Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function f = tomiir(varargin)
0026
0027 %% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     f = getInfo(varargin{3});
0030     return
0031 end
0032
0033 pzm = varargin{1};
0034 if ~isa(pzm, 'pzmodel')
0035     error('### first argument should be a pzmodel.');
```

```

0036 end
0037
0038 if nargin < 1
0039     error('### incorrect number of inputs.');
```

```

0040 end
0041
0042 % Get fs
0043 if nargin == 1
0044     warning('!!! Using default sample rate of 1Hz to design filter.');
```

```

0045     fs = 1;
0046 else
0047     if isa(varargin{2}, 'plist')
0048         pl = varargin{2};
0049         fs = find(pl, 'fs');
```

```

0050         if isempty(fs)
0051             error('### unknown parameter list.');
```

```

0052         end
0053     else
0054         fs = varargin{2};
0055     end
0056 end
0057
0058 % get a and b coefficients
0059 [a,b] = pzmodel2ab(pzm, fs);
0060
0061 % make MIIR filter
0062 f = miir(a,b,fs);
0063 end
0064
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066 % Local Functions %
0067 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0068
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070 %
0071 % FUNCTION: getInfo
0072 %
0073 % DESCRIPTION: Get Info Object
0074 %
0075 % HISTORY: 11-07-07 M Hewitson
0076 % Creation.
0077 %
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079
0080 function ii = getInfo(varargin)
0081 if nargin == 1 && strcmpi(varargin{1}, 'None')
0082     sets = {};
0083     pl = [];
0084 else
0085     sets = {'Default'};
0086     pl = getDefaultPlist;
0087 end
0088 % Build info object
0089 ii = minfo(mfilename, '', '', utils.const.categories.op, '$Id: tomiir.m,v 1.8
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0090 end
0091
0092 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0093 %
0094 % FUNCTION: getDefaultPlist
0095 %

```

```
0096 % DESCRIPTION: Get Default Plist
0097 %
0098 % HISTORY:      11-07-07 M Hewitson
0099 %              Creation.
0100 %
0101 %%%%%%%%%%%
0102
0103 function plo = getDefaultPlist()
0104     plo = plist();
0105 end
0106
```

char

PURPOSE ^

CHAR convert a pzmodel object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a pzmodel object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a pzmodel object into a string.

CALL:      string = char(pzmodel)

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('char')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.8 2008/09/04 15:29:31 ingo Exp $

HISTORY:    02-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [display](#) DISPLAY overloads display functionality for pzmodel objects.
-  [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a pzmodel object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a pzmodel object into a string.
0005 %
0006 % CALL:      string = char(pzmodel)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> pzmodel.getInfo('char')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> pzmodel.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.8 2008/09/04 15:29:31 ingo Exp $
0015 %
```

```

0016 % HISTORY:      02-04-2007 M Hewitson
0017 %
0018 %
0019 %%%%%%%%%%
0020 %
0021 function varargout = char(varargin)
0022 %
0023 %% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'pzmodel');
0030
0031 pstr = '';
0032
0033 for ii = 1:numel(objs)
0034     pstr = [pstr 'pzmodel(' objs(ii).name '), '];
0035 end
0036
0037 varargout{1} = strrep(pstr(1:end-2), '_', '\_');
0038 end
0039
0040 %%%%%%%%%%
0041 %
0042 %%%%%%%%%% Local Functions %%%%%%%%%%
0043 %
0044 %%%%%%%%%%
0045 %
0046 % FUNCTION:      getInfo
0047 %
0048 % DESCRIPTION:   Get Info Object
0049 %
0050 % HISTORY:      11-07-07 M Hewitson
0051 %
0052 %
0053 %%%%%%%%%%
0054 %
0055 function ii = getInfo(varargin)
0056     if nargin == 1 && strcmpi(varargin{1}, 'None')
0057         sets = {};
0058         pl = [];
0059     else
0060         sets = {'Default'};
0061         pl = getDefaultPlist;
0062     end
0063     % Build info object
0064     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.output, '$Id: char.m,v 1.8
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0065 end
0066
0067 %%%%%%%%%%
0068 %
0069 % FUNCTION:      getDefaultPlist
0070 %
0071 % DESCRIPTION:   Get Default Plist
0072 %
0073 % HISTORY:      11-07-07 M Hewitson
0074 %
0075 %
0076 %%%%%%%%%%
0077 %
0078 function plo = getDefaultPlist()
0079     plo = plist();
0080 end
0081

```

display

PURPOSE ^

DISPLAY overloads display functionality for pzmodel objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for pzmodel objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for pzmodel objects.

CALL:      txt      = display(pzmodel)

INPUT:      pzmodel - pole/zero model object

OUTPUT:     txt      - cell array with strings to display the pole/zero model object

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('dieplay')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('display', 'set')

VERSION:    $Id: display.m,v 1.9 2008/09/04 15:29:31 ingo Exp $

HISTORY:    30-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a pzmodel object into a string.

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for pzmodel objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for pzmodel objects.
0005 %
0006 % CALL:      txt      = display(pzmodel)
0007 %
0008 % INPUT:      pzmodel - pole/zero model object
0009 %
0010 % OUTPUT:     txt      - cell array with strings to display the pole/zero model object
```

```

0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 % >> pzmodel.getInfo('dieplay')
0014 %
0015 % Get information about a specified set-plist by calling:
0016 % >> pzmodel.getInfo('display', 'set')
0017 %
0018 % VERSION: $Id: display.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY: 30-01-2007 M Hewitson
0021 % Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027 %%% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 objs = utils.helper.collect_objects(varargin(:), 'pzmodel');
0034
0035 txt = {};
0036
0037 for i=1:numel(objs)
0038     banner = sprintf('---- pzmodel %d ----', i);
0039     txt{end+1} = banner;
0040
0041     % get key and value
0042     name = objs(i).name;
0043     g = objs(i).gain;
0044     ps = objs(i).poles;
0045     zs = objs(i).zeros;
0046     np = length(ps);
0047     nz = length(zs);
0048
0049     % display
0050     txt{end+1} = ['model: ' name];
0051     txt{end+1} = ['gain : ' num2str(g)];
0052     for j=1:np
0053         txt{end+1} = [sprintf('pole %03d: ', j) char(ps(j)) ];
0054     end
0055     for j=1:nz
0056         txt{end+1} = [sprintf('zero %03d: ', j) char(zs(j)) ];
0057     end
0058
0059     banner_end(1:length(banner)) = '-';
0060     txt{end+1} = banner_end;
0061 end
0062
0063 varargout{1} = txt;
0064
0065 if nargout == 0
0066     for ii=1:length(txt)
0067         disp(txt{ii});
0068     end
0069 end
0070
0071 end
0072
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 % Local Functions %
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %
0079 % FUNCTION: getInfo
0080 %
0081 % DESCRIPTION: Get Info Object
0082 %
0083 % HISTORY: 11-07-07 M Hewitson
0084 % Creation.
0085 %
0086 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0087
0088 function ii = getInfo(varargin)
0089 if nargin == 1 && strcmpi(varargin{1}, 'None')
0090     sets = {};
0091     pl = [];
0092 else
0093     sets = {'Default'};
0094     pl = getDefaultPlist;
0095 end
0096 % Build info object

```



```
0097 ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.output, '$Id: display.m,v
1.9 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0098 end
0099
0100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0101 %
0102 % FUNCTION:      getDefaultPlist
0103 %
0104 % DESCRIPTION:  Get Default Plist
0105 %
0106 % HISTORY:      11-07-07 M Hewitson
0107 %                Creation.
0108 %
0109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0110
0111 function plo = getDefaultPlist()
0112     plo = plist();
0113 end
0114
```

fngen

PURPOSE ^

FNGEN creates an arbitrarily long time-series based on the input pzmodel.

SYNOPSIS ^

function varargout = fngen(varargin)

DESCRIPTION ^

```
FNGEN creates an arbitrarily long time-series based on the input pzmodel.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FNGEN creates an arbitrarily long time-series based on the
             input pzmodel.

CALL:       b = fngen(pzm, pl)

PARAMETERS: 'Nsecs' - The number of seconds to produce
              [default: inverse of PSD length]
             'Win'  - The spectral window to use for blending segments
              [default: Kaiser -150dB]

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('fngen')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('fngen', 'none')

VERSION:     $Id: fngen.m,v 1.10 2008/09/05 14:17:33 hewitson Exp $

HISTORY: 12-03-07 M Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input pzmodel.
-  [getupperFreq](#) GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
-  [resp](#) RESP returns the complex response of a pzmodel as an Analysis Object.

This function is called by:

-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input pzmodel.
-  [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % FNGEN creates an arbitrarily long time-series based on the input pzmodel.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: FNGEN creates an arbitrarily long time-series based on the
0005 %                  input pzmodel.
0006 %
0007 % CALL:           b = fngen(pzm, pl)
0008 %
0009 % PARAMETERS:     'Nsecs' - The number of seconds to produce
0010 %                  [default: inverse of PSD length]
0011 %                  'Win'   - The spectral window to use for blending segments
0012 %                  [default: Kaiser -150dB]
0013 %
0014 % M-FILE INFO:    Get information about this methods by calling
0015 %                  >> pzmodel.getInfo('fngen')
0016 %
0017 %                  Get information about a specified set-plist by calling:
0018 %                  >> pzmodel.getInfo('fngen', 'none')
0019 %
0020 % VERSION:        $Id: fngen.m,v 1.10 2008/09/05 14:17:33 hewitson Exp $
0021 %
0022 % HISTORY: 12-03-07 M Hewitson
0023 %                  Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 function varargout = fngen(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 % Collect input variable names
0036 in_names = cell(size(varargin));
0037 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0038
0039 % Collect all AOs and plists
0040 [pzms, pzm_invars] = utils.helper.collect_objects(varargin(:), 'pzmodel', in_names);
0041 pl = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0042
0043 % combine plists
0044 pl = combine(pl, getDefaultPlist());
0045
0046 % Loop over input pzms
0047 bs(1:numel(pzms)) = ao();
0048 for j=1:numel(pzms)
0049     % get sample rate to specify Nyquist
0050     fs = find(pl, 'fs');
0051     if isempty(fs)
0052         fs = round(getupperFreq(pzms(j)) * 10);
0053     end
0054     if fs < 1
0055         fs = 1;
0056     end
0057
0058     % Compute the frequency vector
0059     Nf = 10001;
0060     f1 = 0;
0061     f2 = fs/2;
0062     f = linspace(f1,f2,Nf);
0063     N = 2*(Nf-1);
0064     % Compute model response
0065     w = ao(plist('tsfcn','sqrt(fs/2).*randn(size(t))','fs',fs,'Nsecs',N));
0066     wxx = pwelch(w,plist('Nfft',N,'win',specwin('Rectangular',N)));
0067     axx = resp(pzms(j),plist('f',f));
0068
0069     % Compute desired PSD
0070     wxx.setY(wxx.data.y .* (abs(axx.data.y).^2));
0071
0072     % Call ao/fngen
0073     b = fngen(wxx, pl);
0074     b.setXunits('s');
0075     b.setYunits('');
0076
0077     % Add history
0078     b.addHistory(getInfo, pl, pzm_invars(j), pzms(j).hist);
0079
0080     % Add to outputs
0081     bs(j) = b;
0082 end
0083
0084 % Set outputs
0085 varargout{1} = bs;
0086 end
0087
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089 % Local Functions %

```

```
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0092 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0093 %
0094 % FUNCTION:      getInfo
0095 %
0096 % DESCRIPTION:  Get Info Object
0097 %
0098 % HISTORY:      11-07-07 M Hewitson
0099 %                Creation.
0100 %
0101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103 function ii = getInfo(varargin)
0104     if nargin == 1 && strcmpi(varargin{1}, 'None')
0105         sets = {};
0106         pl   = [];
0107     else
0108         sets = {'Default'};
0109         pl   = getDefaultPlist;
0110     end
0111     % Build info object
0112     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.sigproc, '$Id: fngen.m,v
1.10 2008/09/05 14:17:33 hewitson Exp $', sets, pl);
0113 end
0114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0115 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116 %
0117 % FUNCTION:      getDefaultPlist
0118 %
0119 % DESCRIPTION:  Get Default Plist
0120 %
0121 % HISTORY:      11-07-07 M Hewitson
0122 %                Creation.
0123 %
0124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0125 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0126 function plo = getDefaultPlist()
0127     plo = plist('Nsecs', -1, 'Win', getappdata(0, 'ltpda_default_spectral_window'));
0128 end
0129 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

resp

PURPOSE ^

RESP returns the complex response of a pzmodel as an Analysis Object.

SYNOPSIS ^

function varargout = resp(varargin)

DESCRIPTION ^

```
RESP returns the complex response of a pzmodel as an Analysis Object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RESP returns the complex response of a pzmodel as an Analysis
Object.

CALL:      a = resp(pzm, f);           % compute response for vector f
           a = resp(pzm, f1, f2, nf); % compute response from f1 to f2 in nf
                                           steps.
           a = resp(pzm, pl);          % compute response from parameter list.
           a = resp(pzm);              % compute response

% PARAMETERS: 'f' - a vector of frequencies to evaluate at
              or
              'f1' - start frequency
              'f2' - stop frequency
              'nf' - number of evaluation frequencies
              'scale' - spacing of frequencies: 'lin' or 'log'

              These parameters (one or more) can be defined via a plist,
              with the params keys named in the same way as above

M-FILE INFO: Get information about this methods by calling
              >> pzmodel.getInfo('resp')

              Get information about a specified set-plist by calling:
              >> pzmodel.getInfo('resp', 'Range')

VERSION:     $Id: resp.m,v 1.26 2008/09/04 15:29:31 ingo Exp $

HISTORY:     03-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [getlowerFreq](#) GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
-  [getupperFreq](#) GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
-  [resp](#) RESP returns the complex response of a pzmodel as an Analysis Object.

This function is called by:

-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input pzmodel.
-  [pzmodel](#) PZMODEL constructor for pzmodel class.
-  [resp](#) RESP returns the complex response of a pzmodel as an Analysis Object.
-  [tomfir](#) TOMFIR approximates a pole/zero model with an FIR filter.

SUBFUNCTIONS ^

 `function ii = getInfo(varargin)`
 `function plo = getDefaultPlist(set)`

SOURCE CODE ^

```

0001 % RESP returns the complex response of a pzmodel as an Analysis Object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RESP returns the complex response of a pzmodel as an Analysis
0005 %               Object.
0006 %
0007 % CALL:         a = resp(pzm, f);           % compute response for vector f
0008 %               a = resp(pzm, f1, f2, nf); % compute response from f1 to f2 in nf
0009 %               steps.
0010 %               a = resp(pzm, pl);          % compute response from parameter list.
0011 %               a = resp(pzm);              % compute response
0012 %
0013 % % PARAMETERS: 'f' - a vector of frequencies to evaluate at
0014 %               or
0015 %               'f1' - start frequency
0016 %               'f2' - stop frequency
0017 %               'nf' - number of evaluation frequencies
0018 %               'scale' - spacing of frequencies: 'lin' or 'log'
0019 %
0020 %               These parameters (one or more) can be defined via a plist,
0021 %               with the params keys named in the same way as above
0022 %
0023 % M-FILE INFO: Get information about this methods by calling
0024 %               >> pzmodel.getInfo('resp')
0025 %
0026 %               Get information about a specified set-plist by calling:
0027 %               >> pzmodel.getInfo('resp', 'Range')
0028 %
0029 % VERSION:      $Id: resp.m,v 1.26 2008/09/04 15:29:31 ingo Exp $
0030 %
0031 % HISTORY:      03-04-2007 M Hewitson
0032 %               Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035 %
0036 function varargout = resp(varargin)
0037
0038 %% Check if this is a call for parameters
0039 if utils.helper.isinfocall(varargin{:})
0040     varargout{1} = getInfo(varargin{3});
0041     return
0042 end
0043
0044 %% Input objects checks
0045 if nargin < 1
0046     error('### incorrect number of inputs.')
0047 end
0048
0049 % Collect input pzmodels, plists and input variable names
0050 in_names = cell(size(varargin));
0051 for ii = 1:nargin
0052     in_names{end+1} = inputname(ii);
0053 end
0054 [pzms, invars] = utils.helper.collect_objects(varargin(:), 'pzmodel', in_names);
0055
0056 % Loop over pzmodels
0057 bs = [];
0058 for pp=1: numel(pzms)
0059
0060     % process this pzmodel
0061     pzm = pzms(pp);
0062     % Now look at the model
0063     name = pzm.name;
0064     gain = pzm.gain;
0065     poles = pzm.poles;
0066     zeros = pzm.zeros;
0067     np = length(poles);
0068     nz = length(zeros);
0069
0070     %% check inputs
0071
0072     if nargin == 1
0073         %% a = resp(pzm); %%
0074         % get frequencies for response
0075         minf = getLowerFreq(pzm)/10;
0076         maxf = getUpperFreq(pzm)*10;
0077         pl = plist('f1', minf, 'f2', maxf, 'nf', 1000, 'scale', 'log');

```

```

0078
0079 elseif nargin == 2
0080     if isa(varargin{2}, 'plist')
0081         %%% a = resp(pzm, plist); %%
0082         pl = varargin{2};
0083         if ~isempty(find(pl, 'f'))
0084             pl = combine(pl, getDefaultPlist('List'));
0085         else
0086             pl = combine(pl, getDefaultPlist('Range'));
0087         end
0088     else
0089         %%% a = resp(pzm, f); %%
0090         f = varargin{2};
0091         pl = plist('f',f);
0092     end
0093 elseif nargin == 4
0094     %%% a = resp(pzm, f1, f2, nf); %%
0095     f1 = varargin{2};
0096     f2 = varargin{3};
0097     nf = varargin{4};
0098
0099     pl = plist('f1',f1, 'f2',f2, 'nf',nf, 'scale','lin');
0100
0101 else
0102     error('### incorrect number of inputs.');
```

```

0103 end
0104
0105 % Compute or extract the frequency vector
0106 reshape_f = false;
0107 if ~isempty(find(pl, 'f'))
0108     % Compute from frequency list
0109     f = find(pl, 'f');
0110     % Want to deal with rows
0111     reshape_f = false;
0112     if size(f,1) > 1
0113         f = f.';
0114         reshape_f = true;
0115     end
0116 else
0117     % Compute from frequency range
0118     f1 = find(pl, 'f1');
0119     f2 = find(pl, 'f2');
0120     nf = find(pl, 'nf');
0121     scale = find(pl, 'scale');
0122     switch scale
0123     case 'lin'
0124         f = linspace(f1, f2, nf);
0125     case 'log'
0126         f = logspace(log10(f1), log10(f2), nf);
0127     end
0128 end
0129
0130
0131 % Now compute the response
0132 r = gain;
0133 for j=1:np
0134     if ~isnan(poles(j).f)
0135         [f, pr] = resp(poles(j),f);
0136         r = r .* pr;
0137     end
0138 end
0139 for j=1:nz
0140     if ~isnan(zeros(j).f)
0141         [f, zr] = resp(zeros(j),f);
0142         r = r ./zr;
0143     end
0144 end
0145
0146 % ----- Build output ao
0147 if reshape_f
0148     f = f.';
0149     r = r.';
0150 end
0151
0152 % create new output fsdata
0153 fsd = fsdata(f, r);
0154
0155 % make output analysis object
0156 b = ao(fsd);
0157
0158 % Add history
0159 b.addHistory(getInfo, pl, invars, pzm.hist);
0160
0161 % set name
0162 b.setName(sprintf('resp(%s)', invars{pp}), 'internal');
```

```

0164 % Add to outputs
0165 bs = [bs b];
0166
0167 end
0168
0169 % Outputs
0170 if nargout == 0
0171     iplot(bs)
0172 elseif nargout == 1
0173     varargout{1} = bs;
0174 else
0175     error('### incorrect output arguments');
0176 end
0177
0178 end
0179
0180
0181 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0182 % Local Functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0183 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0184 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0185 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0186 %
0187 % FUNCTION:      getInfo
0188 %
0189 % DESCRIPTION:   Get Info Object
0190 %
0191 % HISTORY:       11-07-07 M Hewitson
0192 %                Creation.
0193 %
0194 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0195
0196 function ii = getInfo(varargin)
0197     if nargin == 1 && strcmpi(varargin{1}, 'None')
0198         sets = {};
0199         pls = [];
0200     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
0201         sets{1} = varargin{1};
0202         pls = getDefaultPlist(sets{1});
0203     else
0204         sets = {'List', 'Range'};
0205         pls = [];
0206         for kk=1:numel(sets)
0207             pls = [pls getDefaultPlist(sets{kk})];
0208         end
0209     end
0210     % Build info object
0211     ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.sigproc, '$Id: resp.m,v
1.26 2008/09/04 15:29:31 ingo Exp $', sets, pls);
0212 end
0213
0214 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0215 %
0216 % FUNCTION:      getDefaultPlist
0217 %
0218 % DESCRIPTION:   Get Default Plist
0219 %
0220 % HISTORY:       11-07-07 M Hewitson
0221 %                Creation.
0222 %
0223 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0224
0225 function plo = getDefaultPlist(set)
0226     switch set
0227     case 'List'
0228         plo = plist('f', []);
0229     case 'Range'
0230         plo = plist('f1', 0.001, 'f2', 1, 'nf', 10000, 'scale', 'lin');
0231     otherwise
0232         plo = plist();
0233     end
0234 end
0235

```


timespan

PURPOSE ^

TIMESPAN timespan object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
TIMESPAN timespan object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION:  TIMESPAN timespan object class constructor.
               Create a timespan object.

SUPER CLASSES: ltpda_uoh < ltpda_uo < ltpda_obj

PROPERTIES:

  Inherit Properties (read only)
    name      - name of object
    created   - creation time (time-object)
    prov      - contains a instance of the provenance class.
    hist      - history of the object (history object)
    version   - cvs-version string.

  Protected Properties (read only)
    startT    - Start time of the time span. (time-object)
    endT      - End time of the time span. (time-object)
    timeformat - The time format of start- AND end- time.
    timezone  - The timezone of start- AND end- time.
    interval  - Interval between start/end time

TIMESPAN Methods:

  Defined Abstract methods:
    char      - returns one character string which represents the object
    copy      - copies an object
    display   - displays an object
    string    - converts an object to a command string which will
                recreate the plist object
    update_struct - updates a object structure to the current tbx-version

  Public methods:

  Protected methods:

  Private methods:
    fromRepository - construct an timespan from a repository
    fromTimespanDef - construct an timespan from start and end time
    computeInterval - compute the interval of the time span

CONSTRUCTORS:
```

```

ts = timespan()
ts = timespan('file_name.mat');
ts = timespan('file_name.xml');
ts = timespan( time, time)
ts = timespan(time, '14:00:05')
ts = timespan('14:00:00', time)
ts = timespan('14:00:00', '14:00:05')
ts = timespan(30000, 50000)
ts = timespan(20000, 30000, 'HH:MM:SS')
ts = timespan(plist)

```

Parameter sets for plist constructor (in order of priority):

From XML File

Construct an TIMESPAN by loading it from an XML file.

'filename' - construct an TIMESPAN from a filename.
 Example: plist('filename', 'ts1.xml')
 [default: empty string]

From MAT File

Construct an TIMESPAN by loading it from a MAT file.

'filename' - construct an TIMESPAN from a filename.
 Example: plist('filename', 'ts1.mat')
 [default: empty string]

From Repository

Construct an TIMESPAN by retrieving it from an LTPDA repository.

'Hostname' - the repository hostname. Only those objects which
 are TIMESPANS are returned.
 [default: 'localhost'];

Additional parameters:

'Database' - The database name [default: 'ltpda']
 'ID' - A vector of object IDs. [default: []]
 'CID' - Retrieve all timespan objects from a particular
 collection.

From Timespan Definition

Construct an TIMESPAN by its properties definition

'startT' - the starting time [default: '1970-01-01 00:30:00.000'];
 'endT' - the ending time [default: '1980-01-01 12:00:00.010'];

The times can be as a time-object, a string or a unix time

Example 1: plist('startT', time, 'endT', '2008-12-31 23:59:59')

Example 2: plist('startT', '2007-12-31 23:59:59', 'endT', 1206442503406)

Additional parameters:

'timezone' - Timezone (string or java object) [default: 'UTC']
 'timeformat' - Time format (string) [default: 'yyyy-mm-dd HH:MM:SS.FFF']

From Plist

'Plist' - construct from a plist. The value passed should be a plist
 object.
 [default: empty plist]

```
M-FILE INFO: The following call returns an minfo object that contains
              information about the timespan constructor:
              >> info = timespan.getInfo
or            >> info = timespan.getInfo('timespan')

              You can get information about class methods by calling:
              >> info = timespan.getInfo(method)
e.g. >> info = timespan.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
              >> info = timespan.getInfo(method, set)
e.g. >> info = timespan.getInfo('timespan', 'None')

VERSION:      $Id: timespan.m,v 1.42 2008/09/07 10:48:46 hewitson Exp $

HISTORY:      03-04-2007 M Hewitson
              Creation

SEE ALSO:      ltpda_uoh, ltpda_uo, ltpda_obj, plist

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [computeInterval](#) COMPUTEINTERVAL compute the interval of the time span.
-  [copy](#) COPY Make copy of time objects depending of the second input
-  [fromTimespanDef](#)
%%%
-  [setEndT](#) SETENDT Set the property 'endT'.
-  [setStartT](#) SETSTARTT Set the property 'startT'.
-  [setTimeformat](#) SETTIMEFORMAT Set the property 'timeformat'.
-  [setTimezone](#) SETTIMEZONE Set the property 'timezone'.
-  [timespan](#) TIMESPAN timespan object class constructor.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [copy](#) COPY Make copy of time objects depending of the second input
-  [timespan](#) TIMESPAN timespan object class constructor.

SUBFUNCTIONS ^

-  [function obj = set.startT\(obj, val\)](#)
-  [function obj = set.endT\(obj, val\)](#)
-  [function obj = set.timeformat\(obj, val\)](#)
-  [function obj = set.timezone\(obj, val\)](#)
-  [function obj = set.interval\(obj, val\)](#)
-  [function obj = timespan\(varargin\)](#)

 function out = VEROUT()
 function ii = getInfo(varargin)
 function out = SETS()
 function out = getDefaultPlist(set)

SOURCE CODE ^

```

0001 % TIMESPAN timespan object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION:  TIMESPAN timespan object class constructor.
0005 %                Create a timespan object.
0006 %
0007 % SUPER CLASSES: ltpda_uoh < ltpda_uo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         name         - name of object
0013 %         created      - creation time (time-object)
0014 %         prov         - contains a instance of the provenance class.
0015 %         hist         - history of the object (history object)
0016 %         version      - cvs-version string.
0017 %
0018 %     Protected Properties (read only)
0019 %         startT        - Start time of the time span. (time-object)
0020 %         endT          - End time of the time span. (time-object)
0021 %         timeformat    - The time format of start- AND end- time.
0022 %         timezone      - The timezone of start- AND end- time.
0023 %         interval      - Interval between start/end time
0024 %
0025 % TIMESPAN Methods:
0026 %
0027 %     Defined Abstract methods:
0028 %         char          - returns one character string which represents the object
0029 %         copy          - copies an object
0030 %         display       - displays an object
0031 %         string        - converts an object to a command string which will
0032 %                        recreate the plist object
0033 %         update_struct - updates a object structure to the current tbx-version
0034 %
0035 %     Public methods:
0036 %
0037 %     Protected methods:
0038 %
0039 %     Private methods:
0040 %         fromRepository - construct an timespan from a repository
0041 %         fromTimespanDef - construct an timespan from start and end time
0042 %         computeInterval - compute the interval of the time span
0043 %
0044 % CONSTRUCTORS:
0045 %
0046 %         ts = timespan()
0047 %         ts = timespan('file_name.mat');
0048 %         ts = timespan('file_name.xml');
0049 %         ts = timespan(time, time)
0050 %         ts = timespan(time, '14:00:05')
0051 %         ts = timespan('14:00:00', time)
0052 %         ts = timespan('14:00:00', '14:00:05')
0053 %         ts = timespan(30000, 50000)
0054 %         ts = timespan(20000, 30000, 'HH:MM:SS')
0055 %         ts = timespan(plist)
0056 %
  
```

```

0057 % Parameter sets for plist constructor (in order of priority):
0058 %
0059 % From XML File
0060 % -----
0061 %
0062 %     Construct an TIMESPAN by loading it from an XML file.
0063 %
0064 %     'filename' - construct an TIMESPAN from a filename.
0065 %                   Example: plist('filename', 'ts1.xml')
0066 %                   [default: empty string]
0067 %
0068 % From MAT File
0069 % -----
0070 %
0071 %     Construct an TIMESPAN by loading it from a MAT file.
0072 %
0073 %     'filename' - construct an TIMESPAN from a filename.
0074 %                   Example: plist('filename', 'ts1.mat')
0075 %                   [default: empty string]
0076 %
0077 % From Repository
0078 % -----
0079 %
0080 %     Construct an TIMESPAN by retrieving it from an LTPDA repository.
0081 %
0082 %     'Hostname' - the repository hostname. Only those objects which
0083 %                   are TIMESPANS are returned.
0084 %                   [default: 'localhost'];
0085 %
0086 %         Additional parameters:
0087 %
0088 %             'Database' - The database name [default: 'ltpda']
0089 %             'ID'       - A vector of object IDs. [default: []]
0090 %             'CID'      - Retrieve all timespan objects from a particular
0091 %                           collection.
0092 %
0093 % From Timespan Definition
0094 % -----
0095 %     Construct an TIMESPAN by its properties definition
0096 %     'startT' - the starting time [default: '1970-01-01 00:30:00.000'];
0097 %     'endT'   - the ending time [default: '1980-01-01 12:00:00.010'];
0098 %                   The times can be as a time-object, a string or a unix time
0099 %                   Example 1: plist('startT', time, 'endT', '2008-12-31 23:59:59')
0100 %                   Example 2: plist('startT', '2007-12-31 23:59:59', 'endT', 1206442503406)
0101 %
0102 %         Additional parameters:
0103 %
0104 %             'timezone' - Timezone (string or java object) [default: 'UTC']
0105 %             'timeformat' - Time format (string) [default: 'yyyy-mm-dd HH:MM:SS.FFF']
0106 %
0107 % From Plist
0108 % -----
0109 %
0110 %     'Plist' - construct from a plist. The value passed should be a plist
0111 %                   object.
0112 %                   [default: empty plist]
0113 %
0114 % M-FILE INFO: The following call returns an minfo object that contains
0115 %                   information about the timespan constructor:
0116 %                   >> info = timespan.getInfo
0117 %                   or >> info = timespan.getInfo('timespan')
0118 %
0119 %                   You can get information about class methods by calling:
0120 %                   >> info = timespan.getInfo(method)
0121 %                   e.g. >> info = timespan.getInfo('eq')
0122 %
0123 %                   You can also restrict the sets of parameters contained in
0124 %                   the minfo object by calling:
0125 %                   >> info = timespan.getInfo(method, set)

```

```

0126 %           e.g. >> info = timespan.getInfo('timespan', 'None')
0127 %
0128 % VERSION:    $Id: timespan.m,v 1.42 2008/09/07 10:48:46 hewitson Exp $
0129 %
0130 % HISTORY:     03-04-2007 M Hewitson
0131 %              Creation
0132 %
0133 % SEE ALSO:    ltpda_uoh, ltpda_uo, ltpda_obj, plist
0134 %
0135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0136 %
0137 classdef timespan < ltpda_uoh
0138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0139 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0140 %              Property definition %
0141 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0142 %
0143 %----- Public (read/write) Properties -----
0144 properties
0145 end
0146 %----- Protected read-only Properties -----
0147 properties (SetAccess = protected)
0148     startT      = time(0);
0149     endT         = time(0);
0150     timeformat   = time(0).timeformat;
0151     timezone     = time(0).timezone;
0152     interval     = '';
0153     version      = '$Id: timespan.m,v 1.42 2008/09/07 10:48:46 hewitson Exp $';
0154 end
0155 %----- Private Properties -----
0156 properties (GetAccess = protected, SetAccess = protected)
0157 end
0158 %----- Abstract Properties -----
0159 properties (Abstract = true, SetAccess = protected)
0160 end
0161 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0162 %              Check property setting %
0163 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0164 %
0165 %
0166 %
0167 %
0168 %
0169 methods
0170     function obj = set.startT(obj, val)
0171         if ~isa(val, 'time')
0172             error('### The value for the property ''startT'' must be a time-object.\nBut it is from the class [%s]',class(val));
0173         end
0174         obj.startT = val;
0175     end
0176     function obj = set.endT(obj, val)
0177         if ~isa(val, 'time')
0178             error('### The value for the property ''endT'' must be a time-object.\nBut it is from the class [%s]',class(val));
0179         end
0180         obj.endT = val;
0181     end
0182     function obj = set.timeformat(obj, val)
0183         if ~ischar(val)
0184             error('### The value for the property ''timeformat'' must be a character string.');
```

```

0195     if ~ischar(val)
0196         error('### The value for the property 'interval' must be a character string.');
```

```

0197     end
0198     obj.interval = val;
0199 end
0200 end
0201
0202 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0203 %                               Constructor                               %
0204 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0205
0206 methods
0207     function obj = timespan(varargin)
0208
0209         import utils.const.*
0210         utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0211
0212         % Call superclass
0213         obj = obj@ltpda_uoh(varargin{:});
0214
0215         % Collect all timespan objects
0216         [ts, invars, rest] = utils.helper.collect_objects(varargin(:), 'timespan');
```

```

0217
0218         if isempty(rest) && ~isempty(ts)
0219             % Do copy constructor and return
0220             utils.helper.msg(msg.OPROC1, 'copy constructor');
```

```

0221             obj = copy(ts, 1);
0222             for kk=1:numel(obj)
0223                 obj(kk).addHistory(timespan.getInfo('timespan', 'None'), [], [], obj(kk).hist);
0224             end
0225             return
0226         end
0227
0228         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Set default values %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0229
0230         switch nargin
0231             case 0
0232                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% no input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0233                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% no input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0234                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% no input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0235                 utils.helper.msg(msg.OPROC1, 'empty constructor');
```

```

0236                 obj.addHistory(timespan.getInfo('timespan', 'None'), [], [], []);
0237
0238             case 1
0239                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% one input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0240                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% one input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0241                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% one input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0242
0243                 if ischar(varargin{1})
0244
0245                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ts = timespan('foo.mat') %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0246                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ts = timespan('foo.xml') %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0247                     utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
0248                     obj = fromFile(obj, varargin{1});
0249
0250                 elseif isa(varargin{1}, 'plist')
0251                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ts = time(plist) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0252
0253                     pl = varargin{1};
0254                     filename = find(pl, 'filename');
```

```

0255                     hostname = find(pl, 'hostname');
```

```

0256                     conn = find(pl, 'conn');
```

```

0257                     ipl = find(pl, 'plist');
```

```

0258                     startT = find(pl, 'startT');
```

```

0259                     endT = find(pl, 'endT');
```

```

0260
0261                     if isempty(startT)
0262                         startT = find(pl, 'start');
```

```

0263                     end

```

```

0264     if isempty(endT)
0265         endT = find(pl, 'end');
0266     end
0267
0268     if ~isempty(filename)
0269         %-----
0270         %--- Construct from file
0271         %-----
0272
0273         utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
0274         obj = fromFile(obj, varargin{1});
0275
0276     elseif ~isempty(hostname) || ~isempty(conn)
0277         %-----
0278         %--- Construct from repository
0279         %-----
0280         utils.helper.msg(msg.OPROC1, 'constructing from repository %s', hostname);
0281         obj = obj.fromRepository(pl);
0282
0283     elseif ~isempty(startT) && ~isempty(endT)
0284         %-----
0285         %--- Construct from start and end times
0286         %-----
0287         utils.helper.msg(msg.OPROC1, 'constructing from start/end times');
0288         obj = obj.fromTimespanDef(pl);
0289
0290     elseif ~isempty(ipl)
0291         %-----
0292         %--- Construct from plist
0293         %-----
0294
0295         if nparams(ipl) == 0
0296             %%% return empty object
0297         else
0298             obj = timespan(ipl);
0299         end
0300
0301     else
0302
0303         if nparams(pl) == 0
0304             %%% if the plist is empty then return the default time object
0305         else
0306             error('### Unknown TIMESPAN constructor method.');

```



```

0333 if (ischar(varargin{1}) || isa(varargin{1}, 'time') || isnumeric(varargin{1})) && ...
0334 (ischar(varargin{2}) || isa(varargin{2}, 'time') || isnumeric(varargin{2}))
0335 %%%%%%%%%%% ts = timespan('14:00:00', '14:00:05') %%%%%%%%%%%
0336 %%%%%%%%%%% ts = timespan('14:00:00', time) %%%%%%%%%%%
0337 %%%%%%%%%%% ts = timespan(time, time) %%%%%%%%%%%
0338 %%%%%%%%%%% ts = timespan(time, '14:00:05') %%%%%%%%%%%
0339
0340 utils.helper.msg(msg.OPROC1, 'constructing from start/end time');
0341 pli = plist('startT', varargin{1}, 'endT', varargin{2});
0342 obj = obj.fromTimespanDef(pli);
0343
0344 elseif isa(varargin{1}, 'database') && isnumeric(varargin{2})
0345 %%%%%%%%%%% f = timespan(database, IDs) %%%%%%%%%%%
0346 utils.helper.msg(msg.OPROC1, 'retrieve from repository');
0347 obj = obj.fromRepository(plist('conn', varargin{1}, 'id', varargin{2}));
0348 elseif isa(varargin{1}, 'timespan') && isa(varargin{2}, 'plist') && isempty(varargin{2}.params)
0349 % pass to copy constructor
0350 obj = timespan(varargin{1});
0351 else
0352 error (' ### Unknown constructor with two inputs. ');
0353 end
0354
0355 case 3
0356 %%%%%%%%%%% three input %%%%%%%%%%%
0357 %%%%%%%%%%% three input %%%%%%%%%%%
0358 %%%%%%%%%%% three input %%%%%%%%%%%
0359
0360 if (ischar(varargin{1}) || isnumeric(varargin{1})) && ...
0361 (ischar(varargin{2}) || isnumeric(varargin{2})) && ...
0362 ischar(varargin{3})
0363 %%%%%%%%%%% obj = timespan('14:00:00', '14:00:05', 'HH:MM:SS') %%%%%%%%%%%
0364 %%%%%%%%%%% obj = timespan( 200000 , 300000 , 'HH:MM:SS') %%%%%%%%%%%
0365
0366 utils.helper.msg(msg.OPROC1, 'constructing from start/end and timeformat');
0367 obj.startT = time(varargin{1}, varargin{3});
0368 obj.endT = time(varargin{2}, varargin{3});
0369 obj.setTimeformat(obj.startT.timeformat, 'internal');
0370
0371 % Compute the interval
0372 obj = computeInterval(obj);
0373
0374 % Add history
0375 ii = timespan.getInfo('timespan', 'From Timespan Definition');
0376 plh = plist(...
0377 'startT', varargin{1}, ...
0378 'endT', varargin{2}, ...
0379 'timeformat', varargin{3});
0380 obj.addHistory(ii, plh, [], []);
0381
0382 else
0383 error (' ### Unknown constructor with three inputs. ');
0384 end
0385
0386 otherwise
0387 [tss, invars, rest] = utils.helper.collect_objects(args, 'timespan');
0388
0389 %%% Do we have a list of AOs as input
0390 if ~isempty(tss) && isempty(rest)
0391 obj = timespan(tss);
0392 else
0393 error('### Unknown number of arguments. ');
0394 end
0395 end
0396
0397 % Consistency check
0398 if obj.startT.utc_epoch_milli > obj.endT.utc_epoch_milli
0399 error('### The start time is larger than the end time. ');
0400 end
0401

```

```

0402     end % End constructor
0403
0404 end
0405
0406 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0407 %                               Methods (public)                               %
0408 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0409 methods
0410     varargout = copy(varargin)
0411     varargout = setStartT(varargin)
0412     varargout = setEndT(varargin)
0413     varargout = setTimeformat(varargin)
0414     varargout = setTimezone(varargin)
0415 end
0416
0417 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0418 %                               Methods (protected)                               %
0419 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0420 methods (Access = protected)
0421 end
0422
0423 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0424 %                               Methods (private)                               %
0425 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0426 methods (Access = private)
0427     obj = fromTimespanDef(obj, pli)
0428     obj = computeInterval(obj)
0429 end
0430
0431 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0432 %                               Methods (static)                               %
0433 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0434 methods (Static)
0435
0436     varargout = update_struct(varargin);
0437
0438     function out = VEROUT()
0439         out = '$Id: timespan.m,v 1.42 2008/09/07 10:48:46 hewitson Exp $';
0440     end
0441
0442     function ii = getInfo(varargin)
0443         ii = utils.helper.generic_getInfo(varargin{:}, 'timespan');
0444     end
0445
0446     function out = SETS()
0447         out = {...
0448             'Default', ...
0449             'From XML File', ...
0450             'From MAT File', ...
0451             'From Repository', ...
0452             'From Timespan Definition', ...
0453             'From Plist'};
0454     end
0455
0456     function out = getDefaultPlist(set)
0457         switch set
0458         case 'Default'
0459             out = plist();
0460         case 'From Repository'
0461             %-- Repository constructor
0462             out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
0463         case 'From Timespan Definition'
0464             %-- From definition constructor
0465             out = plist(...
0466                 'startT', '1970-01-01 00:30:00.000', ...
0467                 'endT', '1980-01-01 12:00:00.010', ...
0468                 'timezone', 'UTC', ...
0469                 'timeformat', 'yyyy-mm-dd HH:MM:SS.FFF');
0470         case 'From XML File'

```

```
0471 %--- Read from XML file
0472 out = plist('filename', '');
0473 case 'From MAT File'
0474 %--- Read from MAT file
0475 out = plist('filename', '');
0476 case 'From Plist'
0477 %--- Create from a plist
0478 out = plist('Plist', []);
0479 otherwise
0480 error('### Unknown set-name [%s] for a default parameter list.', set);
0481 end
0482 end % function out = getDefaultPlist(varargin)
0483
0484 end % End static methods
0485
0486 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0487 %                                Methods (static, private)                                %
0488 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0489
0490 methods (Static, Access=private)
0491 [ao,bo] = abccascade(a1,b1,a2,b2)
0492 end % End static, private methods
0493
0494 end % End classdef
0495
```

setEndT

PURPOSE ^

SETENDT Set the property 'endT'.

SYNOPSIS ^

function varargout = setEndT(varargin)

DESCRIPTION ^

```
SETENDT Set the property 'endT'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'endT'.

CALL:      obj = obj.setEndT('new endT');
           obj = obj.setEndT(plist('endT', 'new endT'));
           obj = setEndT(obj, 'new endT');

INPUTS:    obj - can be a vector, matrix, list, or a mix of them.
           pl  - to set the endT with a plist specify only one plist with
                  only one key-word 'endT'.

M-FILE INFO: Get information about this methods by calling
              >> timespan.getInfo('setEndT')

              Get information about a specified set-plist by calling:
              >> timespan.getInfo('setEndT', 'None')

VERSION:    $Id: setEndT.m,v 1.2 2008/09/04 15:29:31 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [timespan](#) TIMESPAN timespan object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETENDT Set the property 'endT'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'endT'.
0005 %
0006 % CALL:      obj = obj.setEndT('new endT');
0007 %           obj = obj.setEndT(plist('endT', 'new endT'));
0008 %           obj = setEndT(obj, 'new endT');
```

```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl  - to set the endT with a plist specify only one plist with
0012 %                   only one key-word 'endT'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> timespan.getInfo('setEndT')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> timespan.getInfo('setEndT', 'None')
0019 %
0020 % VERSION:     $Id: setEndT.m,v 1.2 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026
0027 function varargout = setEndT(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037
0038     %%% decide whether we modify the timespan-object, or create a new one.
0039     varargin{1} = copy(varargin{1}, nargout);
0040
0041     %%% If the time is a string then convert the string into a time-object.
0042     tt = varargin{2};
0043     if ischar(tt)
0044         tt = time(tt);
0045     end
0046
0047     varargin{1}.endT = tt;
0048     varargin{1}.setTimeformat(tt.timeformat);
0049     objs(ii).computeInterval;
0050     varargout{1} = varargin{1};
0051     return
0052 end
0053
0054 %%% Normal call:
0055 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'timespan');
0056 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0057
0058 endT = rest;
0059
0060 if length(endT) ~=1
0061
0062     if length(pls) == 1
0063         endT = {};
0064         if nparams(pls) == 1 && isparam(pls, 'endT')
0065
0066             %%% If the plist contains only one parameter with the key-word 'endT'
0067             %%% then set 'endT' to the value in the timespan object.
0068             endT{1} = find(pls, 'endT');
0069
0070         else
0071             endT{1} = pls;
0072         end
0073     elseif length(pls) > 1
0074         error('### To set the 'endT' please specify only one plist')
0075     else
0076         error('### Please specify [only one] value.')
0077     end
0078 end
0079
0080 %%% Set the endT
0081 for ii = 1:numel(objs)
0082     %%% decide whether we modify the timespan-object, or create a new one.
0083     objs(ii) = copy(objs(ii), nargout);
0084
0085     %%% If the time is a string then convert the string into a time-object.
0086     tt = varargin{2};
0087     if ischar(tt)
0088         tt = time(tt);
0089     end
0090
0091     objs(ii).endT = tt;
0092     objs(ii).setTimeformat(tt.timeformat);
0093     objs(ii).addHistory(getInfo, plist('endT', tt), [], objs(ii).hist);
0094     objs(ii).computeInterval;

```

```
0095     end
0096
0097     %%% Prepare output
0098     varargout{1} = objs;
0099
0100 end
0101
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103 %                                     Local Functions                                     %
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105
0106 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0107 %
0108 % FUNCTION:      getInfo
0109 %
0110 % DESCRIPTION:  Get Info Object
0111 %
0112 % HISTORY:      11-07-07 M Hewitson
0113 %                Creation.
0114 %
0115 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116
0117 function ii = getInfo(varargin)
0118     if nargin == 1 && strcmpi(varargin{1}, 'None')
0119         sets = {};
0120         pl   = [];
0121     else
0122         sets = {'Default'};
0123         pl   = getDefaultPlist;
0124     end
0125     % Build info object
0126     ii = minfo(mfilename, 'timespan', '', utils.const.categories.helper, '$Id: setEndT.m,v
1.2 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0127 end
0128
0129 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0130 %
0131 % FUNCTION:      getDefaultPlist
0132 %
0133 % DESCRIPTION:  Get Default Plist
0134 %
0135 % HISTORY:      11-07-07 M Hewitson
0136 %                Creation.
0137 %
0138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0139
0140 function plo = getDefaultPlist()
0141     plo = plist('endT', '');
0142 end
0143
```

setStartT

PURPOSE ^

SETSTARTT Set the property 'startT'.

SYNOPSIS ^

function varargout = setStartT(varargin)

DESCRIPTION ^

```
SETSTARTT Set the property 'startT'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'startT'.

CALL:      obj = obj.setStartT('new startT');
           obj = obj.setStartT(plist('startT', 'new startT'));
           obj = setStartT(obj, 'new startT');

INPUTS:    obj - can be a vector, matrix, list, or a mix of them.
           pl  - to set the startT with a plist specify only one plist with
                  only one key-word 'startT'.

M-FILE INFO: Get information about this methods by calling
              >> timespan.getInfo('setStartT')

              Get information about a specified set-plist by calling:
              >> timespan.getInfo('setStartT', 'None')

VERSION:    $Id: setStartT.m,v 1.2 2008/09/04 15:29:31 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [timespan](#) TIMESPAN timespan object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETSTARTT Set the property 'startT'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'startT'.
0005 %
0006 % CALL:      obj = obj.setStartT('new startT');
0007 %           obj = obj.setStartT(plist('startT', 'new startT'));
0008 %           obj = setStartT(obj, 'new startT');
```

```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl  - to set the startT with a plist specify only one plist with
0012 %                   only one key-word 'startT'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> timespan.getInfo('setStartT')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> timespan.getInfo('setStartT', 'None')
0019 %
0020 % VERSION:     $Id: setStartT.m,v 1.2 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026 %
0027 function varargout = setStartT(varargin)
0028 %
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034 %
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037 %
0038 %%% decide whether we modify the timespan-object, or create a new one.
0039 varargin{1} = copy(varargin{1}, nargout);
0040 %
0041 %%% If the time is a string then convert the string into a time-object.
0042 tt = varargin{2};
0043 if ischar(tt)
0044     tt = time(tt);
0045 end
0046 varargin{1}.startT = tt;
0047 varargin{1}.setTimeformat(tt.timeformat);
0048 objs(ii).computeInterval;
0049 varargout{1} = varargin{1};
0050 return
0051 end
0052 %
0053 %%% Normal call:
0054 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'timespan');
0055 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0056 %
0057 startT = rest;
0058 %
0059 if length(startT) ~=1
0060 %
0061     if length(pls) == 1
0062         startT = {};
0063         if nparams(pls) == 1 && isparam(pls, 'startT')
0064 %
0065 %%% If the plist contains only one parameter with the key-word 'startT'
0066 %%% then set 'startT' to the value in the timespan object.
0067 startT{1} = find(pls, 'startT');
0068 %
0069         else
0070             startT{1} = pls;
0071         end
0072     elseif length(pls) > 1
0073         error('### To set the 'startT' please specify only one plist')
0074     else
0075         error('### Please specify [only one] value.')
0076     end
0077 end
0078 %
0079 %%% Set the startT
0080 for ii = 1:numel(objs)
0081 %%% decide whether we modify the timespan-object, or create a new one.
0082     objs(ii) = copy(objs(ii), nargout);
0083 %
0084 %%% If the time is a string then convert the string into a time-object.
0085     tt = varargin{2};
0086     if ischar(tt)
0087         tt = time(tt);
0088     end
0089 %
0090     objs(ii).startT = tt;
0091     objs(ii).setTimeformat(tt.timeformat);
0092     objs(ii).addHistory(getInfo, plist('startT', tt), [], objs(ii).hist);
0093     objs(ii).computeInterval;
0094 end

```



```
0095
0096   %% Prepare output
0097   varargout{1} = objs;
0098
0099 end
0100
0101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0102 %                               Local Functions                               %
0103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104
0105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106 %
0107 % FUNCTION:      getInfo
0108 %
0109 % DESCRIPTION:   Get Info Object
0110 %
0111 % HISTORY:       11-07-07 M Hewitson
0112 %                 Creation.
0113 %
0114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0115
0116 function ii = getInfo(varargin)
0117     if nargin == 1 && strcmpi(varargin{1}, 'None')
0118         sets = {};
0119         pl    = [];
0120     else
0121         sets = {'Default'};
0122         pl    = getDefaultPlist;
0123     end
0124     % Build info object
0125     ii = minfo(mfilename, 'timespan', '', utils.const.categories.helper, '$Id:
setStartT.m,v 1.2 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0126 end
0127
0128 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0129 %
0130 % FUNCTION:      getDefaultPlist
0131 %
0132 % DESCRIPTION:   Get Default Plist
0133 %
0134 % HISTORY:       11-07-07 M Hewitson
0135 %                 Creation.
0136 %
0137 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0138
0139 function plo = getDefaultPlist()
0140     plo = plist('startT', '');
0141 end
0142
```

setTimeformat

PURPOSE ^

SETTIMEFORMAT Set the property 'timeformat'.

SYNOPSIS ^

function varargout = setTimeformat(varargin)

DESCRIPTION ^

```
SETTIMEFORMAT Set the property 'timeformat'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'timeformat'.

CALL:
    obj = obj.setTimeformat('new timeformat');
    obj = obj.setTimeformat(plist('timeformat', 'new timeformat'));
    obj = setTimeformat(obj, 'new timeformat');

INPUTS:
    obj - can be a vector, matrix, list, or a mix of them.
    pl  - to set the timeformat with a plist specify only one plist with
          only one key-word 'timeformat'.

M-FILE INFO: Get information about this methods by calling
    >> timespan.getInfo('setTimeformat')

            Get information about a specified set-plist by calling:
    >> timespan.getInfo('setTimeformat', 'None')

VERSION:
    $Id: setTimeformat.m,v 1.2 2008/09/04 15:29:31 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [timespan](#) TIMESPAN timespan object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETTIMEFORMAT Set the property 'timeformat'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'timeformat'.
0005 %
0006 % CALL:
0007 %     obj = obj.setTimeformat('new timeformat');
0008 %     obj = obj.setTimeformat(plist('timeformat', 'new timeformat'));
0009 %     obj = setTimeformat(obj, 'new timeformat');
```

```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl  - to set the timeformat with a plist specify only one plist with
0012 %                   only one key-word 'timeformat'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> timespan.getInfo('setTimeformat')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> timespan.getInfo('setTimeformat', 'None')
0019 %
0020 % VERSION:     $Id: setTimeformat.m,v 1.2 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%%
0026 %
0027 function varargout = setTimeformat(varargin)
0028 %
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034 %
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037 %
0038 %%% decide whether we modify the timespan-object, or create a new one.
0039 varargin{1} = copy(varargin{1}, nargout);
0040 %
0041 varargin{1}.startT.setTimeformat(varargin{2});
0042 varargin{1}.endT.setTimeformat(varargin{2});
0043 varargin{1}.timeformat = varargin{1}.startT.timeformat;
0044 varargout{1} = varargin{1};
0045 return
0046 end
0047 %
0048 %%% Normal call:
0049 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'timespan');
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051 %
0052 timeformat = rest;
0053 %
0054 if length(timeformat) ~=1
0055 %
0056 if length(pls) == 1
0057     timeformat = {};
0058     if nparams(pls) == 1 && isparam(pls, 'timeformat')
0059 %
0060 %%% If the plist contains only one parameter with the key-word 'timeformat'
0061 %%% then set 'timeformat' to the value in the timespan object.
0062 timeformat{1} = find(pls, 'timeformat');
0063 %
0064 else
0065     timeformat{1} = pls;
0066 end
0067 elseif length(pls) > 1
0068     error('### To set the 'timeformat' please specify only one plist')
0069 else
0070     error('### Please specify [only one] value.')
0071 end
0072 end
0073 %
0074 %%% Set the timeformat
0075 for ii = 1:numel(objs)
0076 %%% decide whether we modify the timespan-object, or create a new one.
0077 objs(ii) = copy(objs(ii), nargout);
0078 %
0079 objs(ii).startT.setTimeformat(timeformat{1});
0080 objs(ii).endT.setTimeformat(timeformat{1});
0081 objs(ii).timeformat = objs(ii).startT.timeformat;
0082 objs(ii).addHistory(getInfo, plist('timeformat', objs(ii).timeformat), [],
0083 objs(ii).hist);
0084 end
0085 %%% Prepare output
0086 varargout{1} = objs;
0087 %
0088 end
0089 %
0090 %%%%%%%%%%%%%%%
0091 % Local Functions
0092 %%%%%%%%%%%%%%%
0093 %

```

```
0094 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0095 %
0096 % FUNCTION:      getInfo
0097 %
0098 % DESCRIPTION:   Get Info Object
0099 %
0100 % HISTORY:       11-07-07 M Hewitson
0101 %                Creation.
0102 %
0103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104
0105 function ii = getInfo(varargin)
0106     if nargin == 1 && strcmpi(varargin{1}, 'None')
0107         sets = {};
0108         pl    = [];
0109     else
0110         sets = {'Default'};
0111         pl    = getDefaultPlist;
0112     end
0113     % Build info object
0114     ii = minfo(mfilename, 'timespan', '', utils.const.categories.helper, '$Id:
setTimeformat.m,v 1.2 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0115 end
0116
0117 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0118 %
0119 % FUNCTION:      getDefaultPlist
0120 %
0121 % DESCRIPTION:   Get Default Plist
0122 %
0123 % HISTORY:       11-07-07 M Hewitson
0124 %                Creation.
0125 %
0126 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0127
0128 function plo = getDefaultPlist()
0129     plo = plist('timeformat', '');
0130 end
0131
```

setTimezone

PURPOSE ^

SETTIMEZONE Set the property 'timezone'.

SYNOPSIS ^

function varargout = setTimezone(varargin)

DESCRIPTION ^

```
SETTIMEZONE Set the property 'timezone'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'timezone'.

CALL:
    obj = obj.setTimezone('new timezone');
    obj = obj.setTimezone(plist('timezone', 'new timezone'));
    obj = setTimezone(obj, 'new timezone');

INPUTS:
    obj - can be a vector, matrix, list, or a mix of them.
    pl - to set the timezone with a plist specify only one plist with
        only one key-word 'timezone'.

M-FILE INFO: Get information about this methods by calling
    >> timespan.getInfo('setTimezone')

    Get information about a specified set-plist by calling:
    >> timespan.getInfo('setTimezone', 'None')

VERSION:      $Id: setTimezone.m,v 1.2 2008/09/04 15:29:31 ingo Exp $

HISTORY:      27-05-2008 Diepholz
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [timespan](#) TIMESPAN timespan object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETTIMEZONE Set the property 'timezone'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'timezone'.
0005 %
0006 % CALL:      obj = obj.setTimezone('new timezone');
0007 %           obj = obj.setTimezone(plist('timezone', 'new timezone'));
0008 %           obj = setTimezone(obj, 'new timezone');
```

```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl  - to set the timezone with a plist specify only one plist with
0012 %                   only one key-word 'timezone'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> timespan.getInfo('setTimezone')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> timespan.getInfo('setTimezone', 'None')
0019 %
0020 % VERSION:     $Id: setTimezone.m,v 1.2 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026 %
0027 function varargout = setTimezone(varargin)
0028 %
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034 %
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037 %
0038 %%% decide whether we modify the timespan-object, or create a new one.
0039 varargin{1} = copy(varargin{1}, nargout);
0040 %
0041 varargin{1}.startT.setTimezone(varargin{2});
0042 varargin{1}.endT.setTimezone(varargin{2});
0043 varargin{1}.timezone = varargin{1}.startT.timezone;
0044 varargout{1} = varargin{1};
0045 return
0046 end
0047 %
0048 %%% Normal call:
0049 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'timespan');
0050 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0051 %
0052 timezone = rest;
0053 %
0054 if length(timezone) ~=1
0055 %
0056     if length(pls) == 1
0057         timezone = {};
0058         if nparams(pls) == 1 && isparam(pls, 'timezone')
0059 %
0060 %%% If the plist contains only one parameter with the key-word 'timezone'
0061 %%% then set 'timezone' to the value in the timespan object.
0062         timezone{1} = find(pls, 'timezone');
0063 %
0064     else
0065         timezone{1} = pls;
0066     end
0067 elseif length(pls) > 1
0068     error('### To set the 'timezone' please specify only one plist')
0069 else
0070     error('### Please specify [only one] value.')
0071 end
0072 end
0073 %
0074 %%% Set the timezone
0075 for ii = 1:numel(objs)
0076 %%% decide whether we modify the timespan-object, or create a new one.
0077     objs(ii) = copy(objs(ii), nargout);
0078 %
0079     objs(ii).startT.setTimezone(timezone{1});
0080     objs(ii).endT.setTimezone(timezone{1});
0081     objs(ii).timezone = objs(ii).startT.timezone;
0082     objs(ii).addHistory(getInfo, plist('timezone', objs(ii).timezone), [],
0083 objs(ii).hist);
0084 end
0085 %%% Prepare output
0086 varargout{1} = objs;
0087 %
0088 end
0089 %
0090 %%%%%%%%%%%%%%
0091 % Local Functions %
0092 %%%%%%%%%%%%%%
0093 %

```

```
0094 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0095 %
0096 % FUNCTION:      getInfo
0097 %
0098 % DESCRIPTION:   Get Info Object
0099 %
0100 % HISTORY:       11-07-07 M Hewitson
0101 %                Creation.
0102 %
0103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104
0105 function ii = getInfo(varargin)
0106     if nargin == 1 && strcmpi(varargin{1}, 'None')
0107         sets = {};
0108         pl    = [];
0109     else
0110         sets = {'Default'};
0111         pl    = getDefaultPlist;
0112     end
0113     % Build info object
0114     ii = minfo(mfilename, 'timespan', '', utils.const.categories.helper, '$Id:
setTimezone.m,v 1.2 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0115 end
0116
0117 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0118 %
0119 % FUNCTION:      getDefaultPlist
0120 %
0121 % DESCRIPTION:   Get Default Plist
0122 %
0123 % HISTORY:       11-07-07 M Hewitson
0124 %                Creation.
0125 %
0126 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0127
0128 function plo = getDefaultPlist()
0129     plo = plist('timezone', '');
0130 end
0131
```

copy

PURPOSE ^

COPY Make copy of time objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of time objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: COPY Make copy of time objects depending of the second input
CALL:      objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
              >> timespan.getInfo('copy')
VERSION:    $Id: copy.m,v 1.6 2008/09/03 16:29:33 hewitson Exp $
HISTORY:    11-07-2008 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of time objects depending of the second input
-  [timespan](#) TIMESPAN timespan object class constructor.

This function is called by:

-  [copy](#) COPY Make copy of time objects depending of the second input
-  [setEndT](#) SETENDT Set the property 'endT'.
-  [setStartT](#) SETSTARTT Set the property 'startT'.
-  [setTimeformat](#) SETTIMEFORMAT Set the property 'timeformat'.
-  [setTimezone](#) SETTIMEZONE Set the property 'timezone'.
-  [timespan](#) TIMESPAN timespan object class constructor.

SOURCE CODE ^

```
0001 % COPY Make copy of time objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of time objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> timespan.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.6 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
```



```
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020
0021 if deepcopy
0022     prov = provenance;
0023     t0 = time;
0024     s = size(old);
0025     obj(s(1),s(2)) = timespan;
0026     for kk=1:numel(old)
0027         utils.helper.msg(utils.const.msg.OPROC2, 'copying timespan');
0028         obj(kk).startT = copy(old(kk).startT, 1);
0029         obj(kk).endT = copy(old(kk).endT, 1);
0030         obj(kk).timeformat = old(kk).timeformat;
0031         obj(kk).timezone = old(kk).timezone;
0032         obj(kk).interval = old(kk).interval;
0033         obj(kk).hist = old(kk).hist;
0034         obj(kk).name = old(kk).name;
0035         %%% reset created time
0036         obj(kk).created = t0;
0037         obj(kk).creator = prov;
0038         obj(kk).version = timespan.VEROUT;
0039     end
0040 else
0041     obj = old;
0042 end
0043 varargout{1} = obj;
0044 end
0045
```

char

PURPOSE ^

CHAR convert a timespan object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a timespan object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a timespan object into a string.

CALL:      string = char(timespan)

M-FILE INFO: Get information about this methods by calling
              >> timespan.getInfo('char')

              Get information about a specified set-plist by calling:
              >> timespan.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.5 2008/09/04 15:29:31 ingo Exp $

HISTORY:    23-07-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a timespan object into a string.

This function is called by:

-  [char](#) CHAR convert a timespan object into a string.
-  [computeInterval](#) COMPUTEINTERVAL compute the interval of the time span.
-  [display](#) DISPLAY overloads display functionality for timespan objects.
-  [string](#) STRING writes a command string that can be used to recreate the input timespan object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a timespan object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a timespan object into a string.
0005 %
0006 % CALL:      string = char(timespan)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> timespan.getInfo('char')
```

```

0010 %
0011 %           Get information about a specified set-plist by calling:
0012 %           >> timespan.getInfo('char', 'set')
0013 %
0014 % VERSION:      $Id: char.m,v 1.5 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:      23-07-2007 Diepholz
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 % Collect all time-objects
0030 objs = utils.helper.collect_objects(varargin(:), 'timespan');
0031
0032 % go through time-object
0033 pstr = '';
0034 for jj=1:numel(objs)
0035     pstr = [pstr '(start: ' char(objs(jj).startT) ' end: ' char(objs(jj).endT) '), '];
0036 end
0037
0038 varargout{1} = pstr(1:end-2);
0039 end
0040
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %                               Local Functions                               %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:      getInfo
0048 %
0049 % DESCRIPTION:   Get Info Object
0050 %
0051 % HISTORY:      11-07-07 M Hewitson
0052 %                Creation.
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'timespan', '', utils.const.categories.output, '$Id: char.m,v
1.5 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:      getDefaultPlist
0071 %
0072 % DESCRIPTION:   Get Default Plist
0073 %
0074 % HISTORY:      11-07-07 M Hewitson
0075 %                Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082

```

display

PURPOSE ^

DISPLAY overloads display functionality for timespan objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for timespan objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for timespan objects.

CALL:      txt = display(ts)

INPUT:     ts - timespan object

OUTPUT:    txt - cell array with strings to display the timespan object

M-FILE INFO: Get information about this methods by calling
              >> timespan.getInfo('display')

              Get information about a specified set-plist by calling:
              >> timespan.getInfo('display', 'None')

VERSION:   $Id: display.m,v 1.9 2008/09/04 15:29:31 ingo Exp $

HISTORY:   23-07-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a timespan object into a string.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for timespan objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for timespan objects.
0005 %
0006 % CALL:      txt = display(ts)
0007 %
0008 % INPUT:     ts - timespan object
0009 %
0010 % OUTPUT:    txt - cell array with strings to display the timespan object
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> timespan.getInfo('display')
```

```

0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> timespan.getInfo('display', 'None')
0017 %
0018 % VERSION:      $Id: display.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY:      23-07-2007 Diepholz
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027 %%% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 % Collect all time-objects
0034 ts = utils.helper.collect_objects(varargin(:), 'timespan');
0035
0036 txt = {};
0037
0038 for ii = 1:numel(ts)
0039     t = ts(ii);
0040
0041     b_banner = sprintf('----- timespan %02d -----', ii);
0042     txt{end+1} = b_banner;
0043     txt{end+1} = ' ';
0044
0045     fields = fieldnames(t);
0046     max_length = length(fields{1});
0047     for jj = 2:length(fields)
0048         if length(fields{jj}) > max_length
0049             max_length = length(fields{jj});
0050         end
0051     end
0052
0053     for jj = 1:length(fields)
0054         field = fields{jj};
0055
0056         str_field = [];
0057         str_field(1:max_length-length(field)) = ' ';
0058         str_field = [field str_field];
0059
0060         % Add before the fields 'created' and 'version' a blank line
0061         if strcmp(field, 'created')
0062             txt{end+1} = ' ';
0063         end
0064
0065         % Display: Number
0066         if isnumeric(t.(field))
0067             txt{end+1} = sprintf('%s: %s', str_field, num2str(t.(field)));
0068
0069         % Display: Strings
0070         elseif ischar(t.(field))
0071             txt{end+1} = sprintf('%s: %s', str_field, t.(field));
0072
0073         % Display: Logicals
0074         elseif islogical(t.(field))
0075             if t.(field) == true
0076                 txt{end+1} = sprintf('%s: true', str_field);
0077             else
0078                 txt{end+1} = sprintf('%s: false', str_field);
0079             end
0080
0081         % Display: Objects
0082         elseif isobject(t.(field))
0083             if isa(t.(field), 'time') || isa(t.(field), 'timeformat') || isa(t.(field),
0084 'plist') || isa(t.(field), 'provenance')
0085                 txt{end+1} = sprintf('%s: %s', str_field, char(t.(field)));
0086             else
0087                 txt{end+1} = sprintf('%s: %s-object', str_field, class(t.(field)));
0088             end
0089
0090         % Display: Java Objects
0091         elseif isjava(t.(field))
0092             if strcmp(class(t.(field)), 'sun.util.calendar.ZoneInfo')
0093                 EE = java.text.SimpleDateFormat('Z');
0094                 EE.setTimeZone(t.(field));
0095                 zone_id = t.(field).getID();
0096                 if strcmp(zone_id, 'UTC')
0097                     txt{end+1} = sprintf('%s: %s', str_field, char(t.(field).getID));
0098                 else
0099                     txt{end+1} = sprintf('%s: %s (UTC%s)', str_field, char(t.(field).getID),

```

```
char (EE.format(ts.start.utc_epoch_milli));
0099     end
0100     else
0101         error ('### Unknown java class %s', class(t.(field)))
0102     end
0103
0104     else
0105         error ('### Please define the output for this property %s', field)
0106     end
0107
0108 end
0109
0110
0111     e_banner(1:length(b_banner)) = '-';
0112     txt{end+1} = e_banner;
0113
0114     txt{end+1} = ' ';
0115     txt{end+1} = ' ';
0116
0117 end
0118
0119 if nargin == 0
0120     for ii=1:length(txt)
0121         disp(txt{ii});
0122     end
0123 end
0124
0125 varargout{1} = txt;
0126
0127 end
0128
0129
0130 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0131 %                                     Local Functions                                     %
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133
0134 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0135 %
0136 % FUNCTION:      getInfo
0137 %
0138 % DESCRIPTION:  Get Info Object
0139 %
0140 % HISTORY:      11-07-07 M Hewitson
0141 %               Creation.
0142 %
0143 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0144
0145 function ii = getInfo(varargin)
0146     if nargin == 1 && strcmpi(varargin{1}, 'None')
0147         sets = {};
0148         pl = [];
0149     else
0150         sets = {'Default'};
0151         pl = getDefaultPlist;
0152     end
0153     % Build info object
0154     ii = minfo(mfilename, 'timespan', '', utils.const.categories.output, '$Id: display.m,v
1.9 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0155 end
0156
0157 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0158 %
0159 % FUNCTION:      getDefaultPlist
0160 %
0161 % DESCRIPTION:  Get Default Plist
0162 %
0163 % HISTORY:      11-07-07 M Hewitson
0164 %               Creation.
0165 %
0166 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0167
0168 function plo = getDefaultPlist()
0169     plo = plist();
0170 end
0171
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input timespan object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input timespan object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate the
input timespan object.

CALL:      cmd = string(obj)

INPUT:     obj - timespan object

OUTPUT:    cmd - command string to create the input object

M-FILE INFO: Get information about this methods by calling
>> timespan.getInfo('string')

            Get information about a specified set-plist by calling:
            >> timespan.getInfo('string', 'None')

VERSION:   $Id: string.m,v 1.5 2008/09/04 15:29:31 ingo Exp $

HISTORY:   15-02-2008 Diepholz
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a timespan object into a string.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input timespan
0002 object.
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input timespan object.
0006 %
0007 % CALL:      cmd = string(obj)
0008 %
0009 % INPUT:     obj - timespan object
0010 %
0011 % OUTPUT:    cmd - command string to create the input object
```

```

0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> timespan.getInfo('string')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> timespan.getInfo('string', 'None')
0018 %
0019 % VERSION: $Id: string.m,v 1.5 2008/09/04 15:29:31 ingo Exp $
0020 %
0021 % HISTORY: 15-02-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 function varargout = string(varargin)
0027 %
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033 %
0034 % Collect all time-objects
0035 objs = utils.helper.collect_objects(varargin(:), 'timespan');
0036 %
0037 cmd = '';
0038 %
0039 for j=1:length(objs)
0040     obj = objs(j);
0041 %
0042     ts_start = obj.startT;
0043     ts_end = obj.endT;
0044     ts_timezone = obj.timezone;
0045     ts_format = obj.timeformat;
0046 %
0047 %%% Convert the java object into a string
0048 ts_timezone = char(ts_timezone.getID);
0049 %
0050 cmd = [cmd 'timespan( plist('startT', ' num2str(ts_start.utc_epoch_milli) ', ' ...
0051 'endT', ' num2str(ts_end.utc_epoch_milli) ', ' ...
0052 'timezone', ' ' ts_timezone ' '
0053 'timeformat', ' ' char(ts_format) ' ' ) ', '];
0054 end
0055 %
0056 %%% Wrap the command only in bracket if the there are more than one object
0057 if length(objs) > 1
0058     cmd = ['[' cmd(1:end-2) '']];
0059 end
0060 %
0061 varargout{1} = cmd;
0062 end
0063 %
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065 % Local Functions %
0066 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0067 %
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION: getInfo
0071 %
0072 % DESCRIPTION: Get Info Object
0073 %
0074 % HISTORY: 11-07-07 M Hewitson
0075 % Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %
0079 function ii = getInfo(varargin)
0080 if nargin == 1 && strcmpi(varargin{1}, 'None')
0081     sets = {};
0082     pl = [];
0083 else
0084     sets = {'Default'};
0085     pl = getDefaultPlist;
0086 end
0087 % Build info object
0088 ii = minfo(mfilename, 'timespan', '', utils.const.categories.output, '$Id: string.m,v
1.5 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0089 end
0090 %
0091 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0092 %
0093 % FUNCTION: getDefaultPlist
0094 %
0095 % DESCRIPTION: Get Default Plist
0096 %

```



```
0097 % HISTORY:      11-07-07 M Hewitson
0098 %
0099 %
0100 %%%%%%%%%%%
0101
0102 function plo = getDefaultPlist()
0103     plo = plist();
0104 end
0105
```

setName

PURPOSE ^

SETNAME Set the property 'name'.

SYNOPSIS ^

function varargout = setName(varargin)

DESCRIPTION ^

```
SETNAME Set the property 'name'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'name'.

CALL:      obj = obj.setName('new name');
           obj = obj.setName(plist('name', 'new name'));
           obj = setName(obj, 'new name');

INPUTS:    obj - can be a vector, matrix, list, or a mix of them.
           pl  - to set the name with a plist specify only one plist with
                  only one key-word 'name'.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('setName')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('setName', 'set')

VERSION:    $Id: setName.m,v 1.11 2008/09/04 15:29:30 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETNAME Set the property 'name'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'name'.
0005 %
0006 % CALL:      obj = obj.setName('new name');
0007 %           obj = obj.setName(plist('name', 'new name'));
0008 %           obj = setName(obj, 'new name');
0009 %
0010 % INPUTS:    obj - can be a vector, matrix, list, or a mix of them.
0011 %           pl  - toset the name with a plist specify only one plist with
```

```

0012 %           only one key-word 'name'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %           >> ao.getInfo('setName')
0016 %
0017 %           Get information about a specified set-plist by calling:
0018 %           >> ao.getInfo('setName', 'set')
0019 %
0020 % VERSION:      $Id: setName.m,v 1.11 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:      27-05-2008 Diepholz
0023 %                Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 function varargout = setName(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037
0038     %%% decide whether we modify the first object, or create a new one.
0039     varargin{1} = copy(varargin{1}, nargout);
0040
0041     varargin{1}.name = varargin{2};
0042     varargout{1} = varargin{1};
0043     return
0044 end
0045
0046 %%% Collect input variable names
0047 in_names = cell(size(varargin));
0048 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0049
0050 [objs, obj_invars, rest] = utils.helper.collect_objects(varargin(:), '', in_names);
0051 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0052
0053 %%% REMARK: Special case for the plist-class because collect_objects collects
0054 %%%         ALL plist-objects even the plist which should set the property.
0055 %%%         In this case must be the plist which sets the property
0056 %%%         at the last position.
0057 if isa(objs, 'plist')
0058     if nparams(objs(end)) == 1 && isparam(objs(end), 'name')
0059         pls = [pls objs(end)];
0060         objs(end) = [];
0061     end
0062 end
0063
0064 %%% If pls contains only one plist with the only key 'name' then set the
0065 %%% property with a plist.
0066 if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'name')
0067     rest{1} = find(pls, 'name');
0068 end
0069
0070 if numel(rest) > 1
0071     error('### Please specify a single name, either in a plist or directly.');
```

```
0098     %% set the value
0099     objs(ii).name = vals;
0100 end
0101
0102     %% Prepare output
0103     varargout{1} = objs;
0104 end
0105
0106
0107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108 %                                     Local Functions                                     %
0109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0110
0111 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0112 %
0113 % FUNCTION:      getInfo
0114 %
0115 % DESCRIPTION:  Get Info Object
0116 %
0117 % HISTORY:      11-07-07 M Hewitson
0118 %               Creation.
0119 %
0120 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0121
0122 function ii = getInfo(varargin)
0123     if nargin == 1 && strcmpi(varargin{1}, 'None')
0124         sets = {};
0125         pl    = [];
0126     else
0127         sets = {'Default'};
0128         pl    = getDefaultPlist;
0129     end
0130     % Build info object
0131     ii = minfo(mfilename, 'ltpda_uo', '', utils.const.categories.helper, '$Id: setName.m,v
1.11 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0132 end
0133
0134 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0135 %
0136 % FUNCTION:      getDefaultPlist
0137 %
0138 % DESCRIPTION:  Get Default Plist
0139 %
0140 % HISTORY:      11-07-07 M Hewitson
0141 %               Creation.
0142 %
0143 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0144
0145 function plo = getDefaultPlist()
0146     plo = plist('name', '');
0147 end
0148
```

append

PURPOSE ^

APPEND append a param-object, plist-object or a key/value pair to the parameter list.

SYNOPSIS ^

function varargout = append(varargin)

DESCRIPTION ^

```
APPEND append a param-object, plist-object or a key/value pair to the parameter list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: APPEND append a param-object, plist-object or a key/value pair to
              the parameter list.

CALL:        pl = append(pl, param-object);
              pl = append(pl, plist-object);
              pl = append(pl, 'key1', 'value1');

              pl = append(pl, combination of the examples above)

REMARK:       It is not possible to append an key/value pair if the key exist
              in the parameter list. Tn this case an error is thrown.

M-FILE INFO:  Get information about this methods by calling
              >> plist.getInfo('append')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('append', 'set')

VERSION:      $Id: append.m,v 1.17 2008/09/04 15:29:31 ingo Exp $

HISTORY:      30-01-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a parameter list into a string.
-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [plist](#) PLIST Plist class object constructor.

SUBFUNCTIONS ^

-  [function add_param\(pl, pp\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % APPEND append a param-object, plist-object or a key/value pair to the parameter list.
```

```
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: APPEND append a param-object, plist-object or a key/value pair to
0005 %               the parameter list.
0006 %
0007 % CALL:         pl = append(pl, param-object);
0008 %               pl = append(pl, plist-object);
0009 %               pl = append(pl, 'key1', 'value1');
0010 %
0011 %               pl = append(pl, combination of the examples above)
0012 %
0013 % REMARK:       It is not possible to append an key/value pair if the key exist
0014 %               in the parameter list. Tn this case an error is thrown.
0015 %
0016 % M-FILE INFO:  Get information about this methods by calling
0017 %               >> plist.getInfo('append')
0018 %
0019 %               Get information about a specified set-plist by calling:
0020 %               >> plist.getInfo('append', 'set')
0021 %
0022 % VERSION:      $Id: append.m,v 1.17 2008/09/04 15:29:31 ingo Exp $
0023 %
0024 % HISTORY:      30-01-07 M Hewitson
0025 %               Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function varargout = append(varargin)
0030
0031     import utils.const.*
0032
0033     %% Check if this is a call for parameters
0034     if utils.helper.isinfocall(varargin{:})
0035         varargout{1} = getInfo(varargin{3});
0036         return
0037     end
0038
0039     utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0040
0041     [objs, invars, rest] = utils.helper.collect_objects(varargin{:}, 'plist');
0042     [pps, invars, rest] = utils.helper.collect_objects(rest{:}, 'param');
0043
0044     %% Decide on a deep copy or a modify
0045     pls = copy(objs, nargout);
0046
0047     %% REMARK: If the rest is an single string and the number of plist is two
0048     %%          then we assume that the rest and the second plist are a key/value
0049     %%          pair.
0050     if numel(rest) == 1 && ischar(rest{1}) && numel(objs) == 2
0051         rest{2} = objs(2);
0052         pls(2) = [];
0053     end
0054
0055     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056     %% First Case: Append plist-objects
0057     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059     pl = pls(1);
0060
0061     %% If we found more than one plist then append the parameters
0062     %% of the second, third, ... plist to the first plist
0063     if numel(pls) > 1
0064         for kk = 2:numel(pls)
0065             for jj = 1:length(pls(kk).params)
0066                 add_param(pl, pls(kk).params(jj));
0067             end
0068         end
0069     end
0070
0071     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072     %% Second Case: Append param-objects
0073     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074
0075     if ~isempty(pps)
0076         for kk = 1:numel(pps)
0077             add_param(pl, pps(kk));
0078         end
0079     end
0080
0081     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082     %% Third Case: Append key/value pairs
0083     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084
0085     if ~isempty(rest)
0086         if mod(numel(rest),2) ~= 0
0087             error('### Please define a ''key'' AND a ''value''%s### to append this pair.',
```

```

char(10));
0088     end
0089     while ~isempty(rest)
0090         key = rest{1};
0091         val = rest{2};
0092
0093         %%% Remove the first two objects from the 'rest' variable
0094         rest(1) = [];
0095         rest(2) = [];
0096
0097         add_param(pl, param(key,val));
0098     end
0099 end
0100
0101 varargout{1} = pl;
0102 end
0103
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %                                     Local Functions
0106 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0110 % DESCRIPTION: The input parameter will only be added if the key doesn't exist
0111 %               in the plist. Throw an error if the key exist in the plist.
0112 function add_param(pl, pp)
0113
0114     for ii = 1:length(pl.params)
0115         if strcmpi(pl.params(ii).key, pp.key)
0116             error('\n### The key [%s] exist in the parameter list.\n### Please use the
function pset.',pp.key);
0117         % pl.params(ii).val = param.val;
0118         return
0119     end
0120 end
0121
0122     utils.helper.msg(utils.const.msg.OPROC1, 'appending parameter %s', pp.key);
0123
0124     % Convert all key characters into upper case characters
0125     pp.setKey(upper(pp.key));
0126     pl.params = [pl.params pp];
0127 end
0128
0129 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0130 %
0131 %
0132 % FUNCTION:      getInfo
0133 %
0134 % DESCRIPTION:  Get Info Object
0135 %
0136 % HISTORY:      11-07-07 M Hewitson
0137 %               Creation.
0138 %
0139 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0140
0141 function ii = getInfo(varargin)
0142     if nargin == 1 && strcmpi(varargin{1}, 'None')
0143         sets = {};
0144         pl = [];
0145     else
0146         sets = {'Default'};
0147         pl = getDefaultPlist;
0148     end
0149     % Build info object
0150     ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: append.m,v
1.17 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0151 end
0152
0153 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0154 %
0155 % FUNCTION:      getDefaultPlist
0156 %
0157 % DESCRIPTION:  Get Default Plist
0158 %
0159 % HISTORY:      11-07-07 M Hewitson
0160 %               Creation.
0161 %
0162 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0163
0164 function plo = getDefaultPlist()
0165     plo = plist();
0166 end
0167

```


combine

PURPOSE ^

COMBINE multiple parameter lists (plist objects) into a single plist.

SYNOPSIS ^

function varargout = combine(varargin)

DESCRIPTION ^

```
COMBINE multiple parameter lists (plist objects) into a single plist.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COMBINE multiple parameter lists (plist objects) into a single
              plist. Duplicate parameters are given priority in the order
              in which they appear in the input.

CALL:        pl = combine(p1, p2, p3);
              pl = combine(p1, [p2 p3], p4)

EXAMPLES:    >> pl1 = plist('A', 1);
              >> pl2 = plist('A', 3);
              >> plo = plist(pl1, pl2)

              Then plo will contain a parameter 'A' with value 1.

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('combine')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('combine', 'set')

VERSION:     $Id: combine.m,v 1.17 2008/09/04 15:29:31 ingo Exp $

HISTORY:     28-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [fromRepository](#) Construct a plist filter from a repository
-  [plist](#) PLIST Plist class object constructor.

SUBFUNCTIONS ^

-  [function pl = add_param\(pl, p\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % COMBINE multiple parameter lists (plist objects) into a single plist.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COMBINE multiple parameter lists (plist objects) into a single
0005 %               plist. Duplicate parameters are given priority in the order
0006 %               in which they appear in the input.
0007 %
0008 % CALL:        pl = combine(p1, p2, p3);
0009 %               pl = combine(p1, [p2 p3], p4)
0010 %
0011 % EXAMPLES:    >> pl1 = plist('A', 1);
0012 %               >> pl2 = plist('A', 3);
0013 %               >> plo = plist(pl1, pl2)
0014 %
0015 %               Then plo will contain a parameter 'A' with value 1.
0016 %
0017 % M-FILE INFO: Get information about this methods by calling
0018 %               >> plist.getInfo('combine')
0019 %
0020 %               Get information about a specified set-plist by calling:
0021 %               >> plist.getInfo('combine', 'set')
0022 %
0023 % VERSION:     $Id: combine.m,v 1.17 2008/09/04 15:29:31 ingo Exp $
0024 %
0025 % HISTORY:     28-04-2007 M Hewitson
0026 %               Creation
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029
0030 function varargout = combine(varargin)
0031
0032 %%% Check if this is a call for parameters
0033 if utils.helper.isinfocall(varargin{:})
0034     varargout{1} = getInfo(varargin{3});
0035     return
0036 end
0037
0038 objs = utils.helper.collect_objects(varargin(:), 'plist');
0039
0040 %%% decide whether we modify the first plist, or create a new one.
0041 pl = copy(objs(1), nargout);
0042
0043 %%% If we found more than one plist then append the parameters
0044 %%% of the second, third, ... plist to the first plist
0045 if numel(objs) > 1
0046     for ii = 2:numel(objs)
0047         % Loop over all params in the current plist
0048         for jj = 1:length(objs(ii).params)
0049             ps = copy(objs(ii).params(jj), 1);
0050             pl = add_param(pl, ps);
0051         end
0052     end
0053 end
0054
0055 varargout{1} = pl;
0056 end
0057
0058 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059 %                               Local Functions
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063 % DESCRIPTION: The input parameter will only be added if the key doesn't exist
0064 %               in the plist.
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066 function pl = add_param(pl, p)
0067
0068 for kk = 1:length(pl.params)
0069     if strcmpi(pl.params(kk).key, p.key)
0070         % If the key exist in pl then do nothing, otherwise add the key value pair.
0071         return
0072     end
0073 end
0074
0075 % Convert all key characters into upper case characters
0076 p.setKey(upper(p.key));
0077
0078 pl.params = [pl.params p];
0079 end
0080
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %
0083 % FUNCTION:     getInfo
0084 %
0085 % DESCRIPTION: Get Info Object
0086 %

```

```
0087 % HISTORY:      11-07-07 M Hewitson
0088 %
0089 %
0090 %%%%%%%%%%
0091
0092 function ii = getInfo(varargin)
0093     if nargin == 1 && strcmpi(varargin{1}, 'None')
0094         sets = {};
0095         pl    = [];
0096     else
0097         sets = {'Default'};
0098         pl    = getDefaultPlist;
0099     end
0100     % Build info object
0101     ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: combine.m,v
1.17 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0102 end
0103
0104 %%%%%%%%%%
0105 %
0106 % FUNCTION:      getDefaultPlist
0107 %
0108 % DESCRIPTION:   Get Default Plist
0109 %
0110 % HISTORY:      11-07-07 M Hewitson
0111 %
0112 %
0113 %%%%%%%%%%
0114
0115 function plo = getDefaultPlist()
0116     plo = plist();
0117 end
0118
```

find

PURPOSE ^

FIND overloads find routine for a parameter list.

SYNOPSIS ^

function varargout = find(varargin)

DESCRIPTION ^

```
FIND overloads find routine for a parameter list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FIND overloads find routine for a parameter list.
              Returns the value corresponding to the first parameters in
              the list with the search-key.

CALL:        a = find(pl, 'key')

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('find')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('find', 'set')

VERSION:     $Id: find.m,v 1.13 2008/09/04 15:29:31 ingo Exp $

HISTORY:     02-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [fromFile](#) Construct a ltpda_ob from a file
-  [fromRepository](#) Construct a plist filter from a repository
-  [plist](#) PLIST Plist class object constructor.
-  [resp](#) RESP shadows miir/iirResp and pzmodel/resp.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % FIND overloads find routine for a parameter list.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FIND overloads find routine for a parameter list.
0005 %              Returns the value corresponding to the first parameters in
```

```

0006 %           the list with the search-key.
0007 %
0008 % CALL:      a = find(pl, 'key')
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %           >> plist.getInfo('find')
0012 %
0013 %           Get information about a specified set-plist by calling:
0014 %           >> plist.getInfo('find', 'set')
0015 %
0016 % VERSION:   $Id: find.m,v 1.13 2008/09/04 15:29:31 ingo Exp $
0017 %
0018 % HISTORY:   02-02-2007 M Hewitson
0019 %           Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 function varargout = find(varargin)
0024
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');
0032
0033 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034 if numel(objs) ~= 1
0035     error('### This function could only work with one plist-object');
0036 end
0037
0038 if numel(rest) ~= 1
0039     error('### Please specify only one ''key''.');
0040 end
0041
0042 if ~ischar(rest{1})
0043     error('### The ''key'' must be a string but it is from the class %s.',
class(rest{1}));
0044 end
0045
0046 pl = objs(1);
0047 key = rest{1};
0048 varargout{1} = [];
0049
0050 for ii=1:length(pl.params)
0051     pp = pl.params(ii);
0052     if strcmpi(pp.key, key)
0053         if isa(pp.val, 'ltpda_obj')
0054             varargout{1} = copy(pp.val, 1);
0055         else
0056             varargout{1} = pp.val;
0057         end
0058     end
0059 end
0060 end
0061
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063 % Local Functions %
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066 %
0067 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0068 %
0069 % FUNCTION:     getInfo
0070 %
0071 % DESCRIPTION:  Get Info Object
0072 %
0073 % HISTORY:     11-07-07 M Hewitson
0074 %             Creation.
0075 %
0076 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077
0078 function ii = getInfo(varargin)
0079 if nargin == 1 && strcmpi(varargin{1}, 'None')
0080     sets = {};
0081     pl = [];
0082 else
0083     sets = {'Default'};
0084     pl = getDefaultPlist;
0085 end
0086 % Build info object
0087 ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: find.m,v 1.13
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0088 end
0089

```

```
0090 %%%%%%%%%%  
0091 %  
0092 % FUNCTION:      getDefaultPlist  
0093 %  
0094 % DESCRIPTION: Get Default Plist  
0095 %  
0096 % HISTORY:        11-07-07 M Hewitson  
0097 %                  Creation.  
0098 %  
0099 %%%%%%%%%%  
0100  
0101 function plo = getDefaultPlist()  
0102     plo = plist();  
0103 end  
0104
```

isparam

PURPOSE ^

ISPARAM look for a given key in the parameter lists.

SYNOPSIS ^

function varargout = isparam(varargin)

DESCRIPTION ^

```
ISPARAM look for a given key in the parameter lists.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: ISPARAM look for a given key in the parameter lists. Exist the key
              in the parameter list then is the result 1 otherwise 0.
              The output size have the same number as the number of the input
              plists.

CALL:        res = isparam(pl, 'key')
              res = isparam(pl1, pl2, 'key')

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('isparam')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('isparam', 'set')

VERSION:     $Id: isparam.m,v 1.9 2008/09/04 15:29:31 ingo Exp $

HISTORY:     07-11-2007 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [plist](#) PLIST Plist class object constructor.

This function is called by:

 [plist](#) PLIST Plist class object constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % ISPARAM look for a given key in the parameter lists.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ISPARAM look for a given key in the parameter lists. Exist the key
0005 %               in the parameter list then is the result 1 otherwise 0.
0006 %               The output size have the same number as the number of the input
0007 %               plists.
0008 %
0009 % CALL:        res = isparam(pl, 'key')
0010 %               res = isparam(pl1, pl2, 'key')
```

```

0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 % >> plist.getInfo('isparam')
0014 %
0015 % Get information about a specified set-plist by calling:
0016 % >> plist.getInfo('isparam', 'set')
0017 %
0018 % VERSION: $Id: isparam.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY: 07-11-2007 Diepholz
0021 % Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = isparam(varargin)
0026
0027 %% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');
0034
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Some plausibility checks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036 if numel(rest) ~= 1
0037     error('### Please specify only one 'key'');
0038 end
0039
0040 if ~ischar(rest{1})
0041     error('### The 'key' must be a string but it is from the class %s.',
0042 class(rest{1}));
0043 end
0044 res = zeros(size(objs));
0045 key = rest{1};
0046
0047 for ii = 1:numel(objs)
0048     pl = objs(ii);
0049     for jj = 1:length(pl.params)
0050         if strcmpi(pl.params(jj).key, key)
0051             res(ii) = 1;
0052             break
0053         end
0054     end
0055 end
0056 end
0057
0058 varargout{1} = res;
0059 end
0060
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062 %
0063 % Local Functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0067 %
0068 % FUNCTION: getInfo
0069 %
0070 % DESCRIPTION: Get Info Object
0071 %
0072 % HISTORY: 11-07-07 M Hewitson
0073 % Creation.
0074 %
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076
0077 function ii = getInfo(varargin)
0078 if nargin == 1 && strcmpi(varargin{1}, 'None')
0079     sets = {};
0080     pl = [];
0081 else
0082     sets = {'Default'};
0083     pl = getDefaultPlist;
0084 end
0085 % Build info object
0086 ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: isparam.m,v
1.9 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0087 end
0088
0089 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090 %
0091 % FUNCTION: getDefaultPlist
0092 %
0093 % DESCRIPTION: Get Default Plist
0094 %

```



```
0095 % HISTORY:      11-07-07 M Hewitson
0096 %
0097 %
0098 %%%%%%%%%%%
0099
0100 function plo = getDefaultPlist()
0101     plo = plist();
0102 end
0103
```

nparams

PURPOSE ^

NPARAMS returns the number of param objects in the list.

SYNOPSIS ^

function varargout = nparams(varargin)

DESCRIPTION ^

```
NPARAMS returns the number of param objects in the list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: NPARAMS returns the number of param objects in the list.

CALL:      n = nparams(plist)

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('nparams')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('nparams', 'set')

VERSION:    $Id: nparams.m,v 1.10 2008/09/04 15:29:31 ingo Exp $

HISTORY:    30-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [plist](#) PLIST Plist class object constructor.

This function is called by:

 [plist](#) PLIST Plist class object constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % NPARAMS returns the number of param objects in the list.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: NPARAMS returns the number of param objects in the list.
0005 %
0006 % CALL:      n = nparams(plist)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> plist.getInfo('nparams')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> plist.getInfo('nparams', 'set')
0013 %
0014 % VERSION:    $Id: nparams.m,v 1.10 2008/09/04 15:29:31 ingo Exp $
```

```
0015 %
0016 % HISTORY:      30-01-2007 M Hewitson
0017 %              Creation
0018 %
0019 %%%%%%%%%%%%%%%
0020
0021 function varargout = nparams(varargin)
0022
0023     %% Check if this is a call for parameters
0024     if utils.helper.isinfocall(varargin{:})
0025         varargout{1} = getInfo(varargin{3});
0026         return
0027     end
0028
0029     objs = utils.helper.collect_objects(varargin(:), 'plist');
0030
0031     n = [];
0032     for ii = 1:numel(objs)
0033         n = [n length(objs(ii).params)];
0034     end
0035
0036     varargout{1} = n;
0037 end
0038
0039 %%%%%%%%%%%%%%%
0040 %              Local Functions
0041 %%%%%%%%%%%%%%%
0042
0043 %%%%%%%%%%%%%%%
0044 %
0045 % FUNCTION:      getInfo
0046 %
0047 % DESCRIPTION:   Get Info Object
0048 %
0049 % HISTORY:      11-07-07 M Hewitson
0050 %              Creation.
0051 %
0052 %%%%%%%%%%%%%%%
0053
0054 function ii = getInfo(varargin)
0055     if nargin == 1 && strcmpi(varargin{1}, 'None')
0056         sets = {};
0057         pl = [];
0058     else
0059         sets = {'Default'};
0060         pl = getDefaultPlist;
0061     end
0062     % Build info object
0063     ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: nparams.m,v
1.10 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0064 end
0065
0066 %%%%%%%%%%%%%%%
0067 %
0068 % FUNCTION:      getDefaultPlist
0069 %
0070 % DESCRIPTION:   Get Default Plist
0071 %
0072 % HISTORY:      11-07-07 M Hewitson
0073 %              Creation.
0074 %
0075 %%%%%%%%%%%%%%%
0076
0077 function plo = getDefaultPlist()
0078     plo = plist();
0079 end
0080
```

pset

PURPOSE ^

PSET set or add a key/value pairor a param-object into the parameter list.

SYNOPSIS ^

function varargout = pset(varargin)

DESCRIPTION ^

```
PSET set or add a key/value pairor a param-object into the parameter list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PSET set or add a key/value pairor a param-object
              into the parameter list. Exist the key in the parameter list
              then becomes the value the new value.

CALL:        pl = pset(pl, param-object);
              pl = pset(pl, key, val);
              pl = pset(pl, key1, val1, key2, val2);

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('pset')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('pset', 'set')

VERSION:      $Id: pset.m,v 1.11 2008/09/04 15:29:31 ingo Exp $

HISTORY:      30-01-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a parameter list into a string.
-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [plist](#) PLIST Plist class object constructor.

SUBFUNCTIONS ^

-  function [add_param\(pl, pp\)](#)
-  function [ii = getInfo\(varargin\)](#)
-  function [plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % PSET set or add a key/value pairor a param-object into the parameter list.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PSET set or add a key/value pairor a param-object
0005 %              into the parameter list. Exist the key in the parameter list
```

```

0006 % then becomes the value the new value.
0007 %
0008 % CALL:      pl = pset(pl, param-object);
0009 %           pl = pset(pl, key, val);
0010 %           pl = pset(pl, key1, val1, key2, val2);
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %               >> plist.getInfo('pset')
0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %               >> plist.getInfo('pset', 'set')
0017 %
0018 % VERSION:    $Id: pset.m,v 1.11 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY:    30-01-07 M Hewitson
0021 %              Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = pset(varargin)
0026
0027     %% Check if this is a call for parameters
0028     if utils.helper.isinfocall(varargin{:})
0029         varargout{1} = getInfo(varargin{3});
0030         return
0031     end
0032
0033     [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');
0034     [pps, invars, rest] = utils.helper.collect_objects(rest(:), 'param');
0035
0036     %% Decide on a deep copy or a modify
0037     pls = copy(objs, nargout);
0038
0039     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040     if (isempty(pps) && isempty(rest)) || mod(numel(rest),2)
0041         error('### Please define a ''key'' AND a ''value''%s### to set this pair.',
0042             char(10));
0043     end
0044
0045     for ii = 1:numel(pls)
0046         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047         if ~isempty(pps)
0048             for jj = 1:numel(pps)
0049                 add_param(pls(ii), pps(jj));
0050             end
0051         end
0052
0053         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054         rest_help = rest;
0055         while ~isempty(rest_help)
0056             key = rest_help{1};
0057             val = rest_help{2};
0058
0059             %% Remove the first two objects from the 'rest_help' variable
0060             rest_help(1) = [];
0061             rest_help(1) = [];
0062
0063             add_param(pls(ii), param(key, val));
0064         end
0065     end
0066
0067     % Set output
0068     if nargout > 0
0069         varargout{1} = pls;
0070     end
0071 end
0072
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 % Local Functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 % DESCRIPTION: check to see if this is one of the parameters we can set,
0079 %               otherwise add it.
0080 function add_param(pl, pp)
0081     found = 0;
0082     for j=1:length(pl.params)
0083         if strcmpi(pl.params(j).key, pp.key)
0084             pl.params(j).setVal(pp.val);
0085             found = 1;
0086             break
0087         end
0088     end
0089     % add this parameter if necessary
0090     if ~found

```

```
0091     pl.params = [pl.params pp];
0092 end
0093 end
0094
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096 %
0097 % FUNCTION:      getInfo
0098 %
0099 % DESCRIPTION:  Get Info Object
0100 %
0101 % HISTORY:      11-07-07 M Hewitson
0102 %               Creation.
0103 %
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105
0106 function ii = getInfo(varargin)
0107     if nargin == 1 && strcmpi(varargin{1}, 'None')
0108         sets = {};
0109         pl = [];
0110     else
0111         sets = {'Default'};
0112         pl = getDefaultPlist;
0113     end
0114     % Build info object
0115     ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: pset.m,v 1.11
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0116 end
0117
0118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0119 %
0120 % FUNCTION:      getDefaultPlist
0121 %
0122 % DESCRIPTION:  Get Default Plist
0123 %
0124 % HISTORY:      11-07-07 M Hewitson
0125 %               Creation.
0126 %
0127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0128
0129 function plo = getDefaultPlist()
0130     plo = plist();
0131 end
0132
```

remove

PURPOSE ^

REMOVE remove a parameter from the parameter list.

SYNOPSIS ^

function varargout = remove(varargin)

DESCRIPTION ^

```
REMOVE remove a parameter from the parameter list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: REMOVE remove a parameter from the parameter list.

CALL:      pl = remove(pl, 2)      - removes 2nd in the list
           pl = remove(pl, 'key') - removes parameter called 'key'

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('remove')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('remove', 'set')

VERSION:    $Id: remove.m,v 1.13 2008/09/04 15:29:31 ingo Exp $

HISTORY:    30-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [plist](#) PLIST Plist class object constructor.

This function is called by:

 [plist](#) PLIST Plist class object constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % REMOVE remove a parameter from the parameter list.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: REMOVE remove a parameter from the parameter list.
0005 %
0006 % CALL:      pl = remove(pl, 2)      - removes 2nd in the list
0007 %           pl = remove(pl, 'key') - removes parameter called 'key'
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> plist.getInfo('remove')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> plist.getInfo('remove', 'set')
```

```

0014 %
0015 % VERSION:      $Id: remove.m,v 1.13 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      30-01-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021 %
0022 function varargout = remove(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');
0031
0032 %%%%%%%%%% Some plausibility checks %%%%%%%%%%
0033 if numel(rest) ~= 1
0034     error('### Specify only an index OR a key-name to remove the corresponding
parameter.');
```

```

0035 end
0036
0037 val = rest{1};
0038
0039 for ii = 1:numel(objs)
0040     pl = objs(ii);
0041     np = length(pl.params);
0042
0043     %%%%%%%%% Remove specified key %%%%%%%%%
0044     if ischar(val)
0045         idx = [];
0046         for i=1:np
0047             key = pl.params(i).key;
0048             if ~strcmpi(key, val)
0049                 idx = [idx i];
0050         end
0051     end
0052     pl.params = pl.params(idx);
0053     if length(idx) == np
0054         warning('!!! No parameter found matching key name in %d. plist', ii);
0055     end
0056
0057     %%%%%%%%% Remove specified position %%%%%%%%%
0058     elseif isnumeric(val)
0059
0060         idx = [1:val-1 val+1:np];
0061         if max(idx) > np
0062             error('### Index exceeds number of parameters in %d. plist', ii);
0063         end
0064         pl.params = pl.params(idx);
0065
0066     %%%%%%%%% Unknown method %%%%%%%%%
0067     else
0068         error('### unknown indexing method')
0069     end
0070
0071     objs(ii) = pl;
0072 end
0073
0074 varargout{1} = objs;
0075 end
0076
0077 %%%%%%%%%%%%%%
0078 %                Local Functions
0079 %%%%%%%%%%%%%%
0080 %
0081 %%%%%%%%%%%%%%
0082 %
0083 % FUNCTION:      getInfo
0084 %
0085 % DESCRIPTION:   Get Info Object
0086 %
0087 % HISTORY:      11-07-07 M Hewitson
0088 %                Creation.
0089 %
0090 %%%%%%%%%%%%%%
0091 %
0092 function ii = getInfo(varargin)
0093     if nargin == 1 && strcmpi(varargin{1}, 'None')
0094         sets = {};
0095         pl = [];
0096     else
0097         sets = {'Default'};
0098         pl = getDefaultPlist;
```



```
0099     end
0100     % Build info object
0101     ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: remove.m,v
1.13 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0102 end
0103
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %
0106 % FUNCTION:      getDefaultPlist
0107 %
0108 % DESCRIPTION:  Get Default Plist
0109 %
0110 % HISTORY:      11-07-07 M Hewitson
0111 %                Creation.
0112 %
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114
0115 function plo = getDefaultPlist()
0116     plo = plist();
0117 end
0118
```

string

PURPOSE ^

STRING converts a plist object to a command string which will recreate the plist object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING converts a plist object to a command string which will recreate the plist object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: STRING converts a plist object to a command string which will
              recreate the plist object.

CALL:        cmd = string(pl)

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('string')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('string', 'set')

VERSION:     $Id: string.m,v 1.21 2008/09/04 15:29:31 ingo Exp $

HISTORY:     29-03-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a parameter list into a string.
-  [plist](#) PLIST Plist class object constructor.
-  [string](#) STRING converts a plist object to a command string which will recreate the plist object.

This function is called by:

-  [plist](#) PLIST Plist class object constructor.
-  [string](#) STRING converts a plist object to a command string which will recreate the plist object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING converts a plist object to a command string which will recreate the plist
0002 object.
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 % DESCRIPTION: STRING converts a plist object to a command string which will
0005 %               recreate the plist object.
0006 %
0007 % CALL:        cmd = string(pl)
```

```

0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 % >> plist.getInfo('string')
0011 %
0012 % Get information about a specified set-plist by calling:
0013 % >> plist.getInfo('string', 'set')
0014 %
0015 % VERSION: $Id: string.m,v 1.21 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY: 29-03-2007 M Hewitson
0018 % Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = string(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 objs = utils.helper.collect_objects(varargin(:), 'plist');
0031
0032 pstr = '';
0033
0034 for ii = 1:numel(objs)
0035     pl = objs(ii);
0036
0037     if ~isempty(pl.params)
0038         pstr = [pstr 'plist('];
0039         for j=1:length(pl.params)
0040             p = pl.params(j);
0041             if ischar(p.val)
0042                 pstr = [pstr ' ' p.key ' ', ' ' char(p.val) ' ', '];
0043             elseif isnumeric(p.val)
0044                 pstr = [pstr ' ' p.key ' ', ' ' utils.helper.mat2str(p.val) ' ', '];
0045             elseif isa(p.val, 'ao')
0046                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0047             elseif isa(p.val, 'history')
0048                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0049             elseif isa(p.val, 'miir')
0050                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0051             elseif isa(p.val, 'mfir')
0052                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0053             elseif isa(p.val, 'specwin')
0054                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0055             elseif isa(p.val, 'pz') || isa(p.val, 'pole') || isa(p.val, 'zero') ||
isa(p.val, 'pzmodel')
0056                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0057             elseif isa(p.val, 'time')
0058                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0059             elseif isa(p.val, 'plist')
0060                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0061             elseif isa(p.val, 'timeformat')
0062                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0063             elseif isa(p.val, 'timespan')
0064                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0065             elseif isa(p.val, 'param')
0066                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0067             elseif isa(p.val, 'provenance')
0068                 pstr = [pstr ' ' p.key ' ', ' ' string(p.val) ' ', '];
0069             elseif iscell(p.val)
0070                 val_str = ['{ '];
0071                 for ii = 1:numel(p.val)
0072                     cell_val = p.val{ii};
0073                     if ischar(cell_val)
0074                         val_str = [val_str ' ' p.val{ii} ' ', '];
0075                     elseif isnumeric(cell_val)
0076                         val_str = [val_str utils.helper.mat2str(p.val{ii}) ' ', '];
0077                     elseif isobject(cell_val)
0078                         val_str = [val_str string(p.val{ii}) ' ', '];
0079                     else
0080                         error('### Unknown object in cell array');
0081                     end
0082                 end
0083                 val_str = val_str(1:end-2);
0084                 val_str = [val_str ' }'];
0085                 pstr = [pstr ' ' p.key ' ', ' ' val_str ' ', '];
0086             elseif isa(p.val, 'cdata') || isa(p.val, 'fsdata') || ...
isa(p.val, 'tsdata') || isa(p.val, 'xydata') || ...
isa(p.val, 'xyzdata')
0089                 pstr = [pstr ' ' p.key ' ', ' ' class(p.val) '(', ' ');
0090             else
0091                 error('!!! Can not handle the class [%s].', class(p.val));
0092             end

```

```
0093         end
0094
0095         % close bracket
0096         if ~isempty(pstr)
0097             pstr = pstr(1:end-2);
0098         end
0099         pstr = [pstr '), '];
0100     else
0101         pstr = [pstr 'plist(), '];
0102     end
0103 end
0104
0105 if numel(objs) > 1
0106     pstr = ['[' pstr(1:end-2) ' ]'];
0107 else
0108     pstr = pstr(1:end-2);
0109 end
0110
0111 varargout{1} = pstr;
0112 end
0113
0114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0115 %                                     Local Functions
0116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0117
0118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0119 %
0120 % FUNCTION:      getInfo
0121 %
0122 % DESCRIPTION:  Get Info Object
0123 %
0124 % HISTORY:      11-07-07 M Hewitson
0125 %               Creation.
0126 %
0127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0128
0129 function ii = getInfo(varargin)
0130     if nargin == 1 && strcmpi(varargin{1}, 'None')
0131         sets = {};
0132         pl = [];
0133     else
0134         sets = {'Default'};
0135         pl = getDefaultPlist;
0136     end
0137     % Build info object
0138     ii = minfo(mfilename, 'plist', '', utils.const.categories.helper, '$Id: string.m,v
1.21 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0139 end
0140
0141 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0142 %
0143 % FUNCTION:      getDefaultPlist
0144 %
0145 % DESCRIPTION:  Get Default Plist
0146 %
0147 % HISTORY:      11-07-07 M Hewitson
0148 %               Creation.
0149 %
0150 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0151
0152 function plo = getDefaultPlist()
0153     plo = plist();
0154 end
0155
```

copy

PURPOSE ^

COPY Make copy of plist objects depending of the second input

SYNOPSIS ^

function varargout = **copy**(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of plist objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of plist objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('copy')

VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [append](#) APPEND append a param-object, plist-object or a key/value pair to the parameter list.
-  [combine](#) COMBINE multiple parameter lists (plist objects) into a single plist.
-  [copy](#) COPY Make copy of plist objects depending of the second input
-  [find](#) FIND overloads find routine for a parameter list.
-  [plist](#) PLIST Plist class object constructor.
-  [pset](#) PSET set or add a key/value pair a param-object into the parameter list.

SOURCE CODE ^

```
0001 % COPY Make copy of plist objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of plist objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> plist.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
```

```
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         % Reset these values on copy
0022         prov = provenance;
0023         t0 = time;
0024         s = size(old);
0025         obj(s(1),s(2)) = plist;
0026         for kk=1:numel(old)
0027             utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
0028             if ~isempty(old(kk).params)
0029                 obj(kk).params = copy(old(kk).params, 1);
0030             end
0031             obj(kk).name = old(kk).name;
0032             obj(kk).created = t0;
0033             obj(kk).creator = prov;
0034             obj(kk).version = plist.VEROUT;
0035         end
0036     else
0037         obj = old;
0038     end
0039     varargout{1} = obj;
0040 end
0041
```

char

PURPOSE ^

CHAR convert a parameter list into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a parameter list into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a parameter list into a string.

CALL:      string = char(plist);

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('char')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.21 2008/09/04 15:29:31 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a parameter list into a string.
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [append](#) APPEND append a param-object, plist-object or a key/value pair to the parameter list.
-  [char](#) CHAR convert a parameter list into a string.
-  [plist](#) PLIST Plist class object constructor.
-  [pset](#) PSET set or add a key/value pair or a param-object into the parameter list.
-  [string](#) STRING converts a plist object to a command string which will recreate the plist object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a parameter list into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a parameter list into a string.
0005 %
```

```

0006 % CALL:      string = char(plist);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %             >> plist.getInfo('char')
0010 %
0011 %             Get information about a specified set-plist by calling:
0012 %             >> plist.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.21 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:    04-02-2007 M Hewitson
0017 %             Creation
0018 %
0019 %%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'plist');
0030
0031 pstr = '';
0032 for ii = 1:numel(objs)
0033     pl = objs(ii);
0034     if ~isempty(pl.params)
0035         pstr = [pstr '(' char(pl.params) '), '];
0036     else
0037         pstr = [pstr '(empty-plist), '];
0038     end
0039 end
0040
0041 if length(pstr)>1
0042     pstr = pstr(1:end-2);
0043 else
0044     pstr = '(empty-plist)';
0045 end
0046
0047 %%% Prepare output
0048 varargout{1} = pstr;
0049 end
0050
0051 %%%%%%%%%%%
0052 %                               Local Functions
0053 %%%%%%%%%%%
0054
0055 %%%%%%%%%%%
0056 %
0057 % FUNCTION:      getInfo
0058 %
0059 % DESCRIPTION:   Get Info Object
0060 %
0061 % HISTORY:      11-07-07 M Hewitson
0062 %              Creation.
0063 %
0064 %%%%%%%%%%%
0065
0066 function ii = getInfo(varargin)
0067     if nargin == 1 && strcmpi(varargin{1}, 'None')
0068         sets = {};
0069         pl = [];
0070     else
0071         sets = {'Default'};
0072         pl = getDefaultPlist;
0073     end
0074     % Build info object
0075     ii = minfo(mfilename, 'plist', '', utils.const.categories.output, '$Id: char.m,v 1.21
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0076 end
0077
0078 %%%%%%%%%%%
0079 %
0080 % FUNCTION:      getDefaultPlist
0081 %
0082 % DESCRIPTION:   Get Default Plist
0083 %
0084 % HISTORY:      11-07-07 M Hewitson
0085 %              Creation.
0086 %
0087 %%%%%%%%%%%
0088
0089 function plo = getDefaultPlist()
0090     plo = plist();

```


0091 end

0092

display

PURPOSE ^

DISPLAY display plist object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY display plist object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY display plist object.

M-FILE INFO: Get information about this methods by calling
              >> plist.getInfo('display')

              Get information about a specified set-plist by calling:
              >> plist.getInfo('display', 'set')

VERSION:      $Id: display.m,v 1.11 2008/09/04 15:29:31 ingo Exp $

HISTORY:      30-01-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [display](#) DISPLAY display plist object.
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [display](#) DISPLAY display plist object.
-  [plist](#) PLIST Plist class object constructor.

SUBFUNCTIONS ^

-  function new_txt = single_cell(txt_field)
-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % DISPLAY display plist object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY display plist object.
0005 %
0006 % M-FILE INFO: Get information about this methods by calling
0007 %               >> plist.getInfo('display')
0008 %
0009 %               Get information about a specified set-plist by calling:
0010 %               >> plist.getInfo('display', 'set')
0011 %
```

```

0012 % VERSION:      $Id: display.m,v 1.11 2008/09/04 15:29:31 ingo Exp $
0013 %
0014 % HISTORY:       30-01-2007 M Hewitson
0015 %               Creation
0016 %
0017 %%%%%%%%%%%%%%%
0018
0019 function varargout = display(varargin)
0020
0021 %% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 objs = utils.helper.collect_objects(varargin(:), 'plist');
0028
0029 txt = {};
0030
0031 for ii=1:numel(objs)
0032     pl = objs(ii);
0033     n = length(pl.params);
0034
0035     banner_start = sprintf('----- plist %02d -----', ii);
0036     txt{end+1} = banner_start;
0037
0038     txt{end+1} = sprintf('n params: %d', n);
0039     params = pl.params;
0040     if n>0
0041         txt{end+1} = display(params);
0042     end
0043
0044     txt = single_cell(txt);
0045
0046     banner_end(1:length(banner_start)) = '-';
0047     txt{end+1} = banner_end;
0048
0049 end
0050
0051 if nargout == 0
0052     for ii=1:length(txt)
0053         disp(txt{ii});
0054     end
0055 end
0056
0057 varargout{1} = txt;
0058 end
0059
0060 %%%%%%%%%%%%%%%
0061 %               Local Functions
0062 %%%%%%%%%%%%%%%
0063
0064 function new_txt = single_cell(txt_field)
0065
0066 new_txt = {};
0067 for ii=1:length(txt_field)
0068     if iscell(txt_field{ii})
0069         hh = single_cell(txt_field{ii});
0070         new_txt(end+1:end+length(hh)) = hh(1:end);
0071     else
0072         new_txt{end+1} = txt_field{ii};
0073     end
0074 end
0075 end
0076
0077 %%%%%%%%%%%%%%%
0078 %
0079 % FUNCTION:      getInfo
0080 %
0081 % DESCRIPTION:   Get Info Object
0082 %
0083 % HISTORY:       11-07-07 M Hewitson
0084 %               Creation.
0085 %
0086 %%%%%%%%%%%%%%%
0087
0088 function ii = getInfo(varargin)
0089 if nargin == 1 && strcmpi(varargin{1}, 'None')
0090     sets = {};
0091     pl = [];
0092 else
0093     sets = {'Default'};
0094     pl = getDefaultPlist;
0095 end
0096 % Build info object
0097 ii = minfo(mfilename, 'plist', '', utils.const.categories.output, '$Id: display.m,v

```

```
1.11 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0098 end
0099
0100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0101 %
0102 % FUNCTION:      getDefaultPlist
0103 %
0104 % DESCRIPTION:  Get Default Plist
0105 %
0106 % HISTORY:      11-07-07 M Hewitson
0107 %                Creation.
0108 %
0109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0110
0111 function plo = getDefaultPlist()
0112     plo = plist();
0113 end
0114
```

resp

PURPOSE ^

RESP shadows miir/iirResp and pzmodel/resp.

SYNOPSIS ^

function varargout = resp(varargin)

DESCRIPTION ^

```
RESP shadows miir/iirResp and pzmodel/resp.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RESP shadows miir/resp and pzmodel/resp.
The following parameters are looked for in this order:
    'filter' - miir/resp or mfir/resp is called
    'pzmodel' - pzmodel/resp is called

CALL:
    resp(plist);
    filt_resp = resp(plist);

M-FILE INFO: Get information about this methods by calling
    >> plist.getInfo('resp')
             Get information about a specified set-plist by calling:
    >> plist.getInfo('resp', 'set')

VERSION:      $Id: resp.m,v 1.15 2008/09/04 15:29:31 ingo Exp $

HISTORY:      12-04-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [find](#) FIND overloads find routine for a parameter list.
-  [plist](#) PLIST Plist class object constructor.
-  [resp](#) RESP shadows miir/iirResp and pzmodel/resp.

This function is called by:

-  [plist](#) PLIST Plist class object constructor.
-  [resp](#) RESP shadows miir/iirResp and pzmodel/resp.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % RESP shadows miir/iirResp and pzmodel/resp.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: RESP shadows miir/resp and pzmodel/resp.
0005 %
0006 % The following parameters are looked for in this order:
0007 %
0008 % 'filter' - miir/resp or mfir/resp is called
0009 % 'pzmodel' - pzmodel/resp is called
0010 %
0011 % CALL:          resp(plist);
0012 %               filt_resp = resp(plist);
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> plist.getInfo('resp')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> plist.getInfo('resp', 'set')
0019 %
0020 % VERSION:      $Id: resp.m,v 1.15 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:      12-04-2007 M Hewitson
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 function varargout = resp(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 objs = utils.helper.collect_objects(varargin(:), 'plist');
0036
0037 b = [];
0038
0039 for ii = 1:numel(objs)
0040
0041     pl = objs(ii);
0042
0043     filt = find(pl, 'filter');
0044     pzm = find(pl, 'pzmodel');
0045     if ~isempty(filt)
0046         if isa(filt, 'miir') || isa(filt, 'mfir')
0047             b = [b resp(filt, pl)];
0048         else
0049             error('### The %d. input parameter list contains an unknown filter object.',
0050 ii);
0051         end
0052     elseif ~isempty(pzm)
0053         if isa(pzm, 'pzmodel')
0054             b = [b resp(pzm, pl)];
0055         else
0056             error('### Unknown pzmodel object in %d. plist-object.', ii);
0057         end
0058     else
0059         error('### The %d. input parameter list must contain a ''filter'' or a
0060 ''pzmodel'' parameter.', ii);
0061     end
0062
0063 %%% Define Outputs
0064 if nargout == 0
0065     iplot(b)
0066 elseif nargout == 1
0067     varargout{1} = b;
0068 else
0069     error('### Incorrect number of output arguments');
0070 end
0071 end
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073 % Local Functions %
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075
0076 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077 %
0078 % FUNCTION:      getInfo
0079 %
0080 % DESCRIPTION: Get Info Object
0081 %
0082 % HISTORY:      11-07-07 M Hewitson
0083 %               Creation.
0084 %
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086
0087 function ii = getInfo(varargin)

```

```
0088     if nargin == 1 && strcmpi(varargin{1}, 'None')
0089         sets = {};
0090         pl    = [];
0091     else
0092         sets = {'Default'};
0093         pl    = getDefaultPlist;
0094     end
0095     % Build info object
0096     ii = minfo(mfilename, 'plist', '', utils.const.categories.sigproc, '$Id: resp.m,v 1.15
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0097 end
0098
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100 %
0101 % FUNCTION:      getDefaultPlist
0102 %
0103 % DESCRIPTION:  Get Default Plist
0104 %
0105 % HISTORY:      11-07-07 M Hewitson
0106 %               Creation.
0107 %
0108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109
0110 function plo = getDefaultPlist()
0111     pl = plist('type', 'lowpass');
0112     plo = plist('filter', mfir(pl));
0113 end
0114
```

copy

PURPOSE ^

COPY Make copy of specwin objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of specwin objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of specwin objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> specwin.getInfo('copy')

VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [specwin](#) % SPECWIN spectral window object class constructor.

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 % COPY Make copy of specwin objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of specwin objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> specwin.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('specwin');
0022         s = size(old);
0023         obj(s(1),s(2)) = specwin;
0024         for kk=1: numel(old)
0025             utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).type);
```



```
0026     for jj=1:length(ps)
0027         obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028     end
0029     % reset on copy
0030     obj(kk).version = specwin.VEROUT;
0031 end
0032 else
0033     obj = old;
0034 end
0035 varargout{1} = obj;
0036 end
0037
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

char

PURPOSE ^

CHAR convert a specwin object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a specwin object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a specwin object into a string.

CALL:      string = char(sw);

M-FILE INFO: Get information about this methods by calling
              >> specwin.getInfo('char')

              Get information about a specified set-plist by calling:
              >> specwin.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.7 2008/09/04 15:29:31 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % CHAR convert a specwin object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a specwin object into a string.
0005 %
0006 % CALL:      string = char(sw);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> specwin.getInfo('char')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> specwin.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.7 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:    04-02-2007 M Hewitson
0017 %              Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
```

```

0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 %%% Get input
0030 objs = utils.helper.collect_objects(varargin(:), 'specwin');
0031
0032 pstr = [];
0033 for kk = 1:numel(objs)
0034     w = objs(kk);
0035
0036     pstr = ['(' pstr w.type];
0037     pstr = [pstr ', alpha=' num2str(w.alpha)];
0038     pstr = [pstr ', psll=' num2str(w.psll)];
0039     pstr = [pstr ', rov=' num2str(w.rov)];
0040     pstr = [pstr ', nenbw=' num2str(w.nenbw)];
0041     pstr = [pstr ', w3db=' num2str(w.w3db)];
0042     pstr = [pstr ', flatness=' num2str(w.flatness) '), '];
0043
0044     pstr = strrep(pstr, '_', '\_');
0045 end
0046
0047 varargout{1} = pstr(1:end-2);
0048
0049 end
0050
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052 %                                     Local Functions
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056 %
0057 % FUNCTION:      getInfo
0058 %
0059 % DESCRIPTION:   Get Info Object
0060 %
0061 % HISTORY:       11-07-07 M Hewitson
0062 %                 Creation.
0063 %
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065
0066 function ii = getInfo(varargin)
0067     if nargin == 1 && strcmpi(varargin{1}, 'None')
0068         sets = {};
0069         pl = [];
0070     else
0071         sets = {'Default'};
0072         pl = getDefaultPlist;
0073     end
0074     % Build info object
0075     ii = minfo(mfilename, 'specwin', '', utils.const.categories.output, '$Id: char.m,v 1.7
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0076 end
0077
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079 %
0080 % FUNCTION:      getDefaultPlist
0081 %
0082 % DESCRIPTION:   Get Default Plist
0083 %
0084 % HISTORY:       11-07-07 M Hewitson
0085 %                 Creation.
0086 %
0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088
0089 function plo = getDefaultPlist()
0090     plo = plist();
0091 end
0092

```

display

PURPOSE ^

DISPLAY overloads display functionality for specwin objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for specwin objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for specwin objects.

CALL:      txt      = display(specwin)

INPUT:      specwin - spectral window object

OUTPUT:     txt      - cell array with strings to display the spectral window object

M-FILE INFO: Get information about this methods by calling
              >> specwin.getInfo('display')

              Get information about a specified set-plist by calling:
              >> specwin.getInfo('display', 'set')

VERSION:    $Id: display.m,v 1.10 2008/09/04 15:29:31 ingo Exp $

HISTORY:     30-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for specwin objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for specwin objects.
0005 %
0006 % CALL:      txt      = display(specwin)
0007 %
0008 % INPUT:      specwin - spectral window object
0009 %
0010 % OUTPUT:     txt      - cell array with strings to display the spectral window object
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> specwin.getInfo('display')
0014 %
0015 %              Get information about a specified set-plist by calling:
0016 %              >> specwin.getInfo('display', 'set')
```

```
0017 %
0018 % VERSION:      $Id: display.m,v 1.10 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY:      30-01-2007 M Hewitson
0021 %               Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024 %
0025 function varargout = display(varargin)
0026 %
0027 %%% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032 %
0033 % Get specwin objects
0034 objs = utils.helper.collect_objects(varargin(:), 'specwin');
0035 %
0036 % get display text
0037 txt = utils.helper.objdisp(objs);
0038 %
0039 if nargout == 0
0040     for ii=1:length(txt)
0041         disp(txt{ii});
0042     end
0043 else
0044     varargout{1} = txt;
0045 end
0046 %
0047 end
0048 %
0049 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0050 %                               Local Functions                               %
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052 %
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054 %
0055 % FUNCTION:      getInfo
0056 %
0057 % DESCRIPTION:   Get Info Object
0058 %
0059 % HISTORY:      11-07-07 M Hewitson
0060 %               Creation.
0061 %
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063 %
0064 function ii = getInfo(varargin)
0065     if nargin == 1 && strcmpi(varargin{1}, 'None')
0066         sets = {};
0067         pl = [];
0068     else
0069         sets = {'Default'};
0070         pl = getDefaultPlist;
0071     end
0072     % Build info object
0073     ii = minfo(mfilename, 'specwin', '', utils.const.categories.output, '$Id: display.m,v
1.10 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0074 end
0075 %
0076 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077 %
0078 % FUNCTION:      getDefaultPlist
0079 %
0080 % DESCRIPTION:   Get Default Plist
0081 %
0082 % HISTORY:      11-07-07 M Hewitson
0083 %               Creation.
0084 %
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086 %
0087 function plo = getDefaultPlist()
0088     plo = plist();
0089 end
0090 %
```

plot

PURPOSE ^

PLOT plots a specwin object.

SYNOPSIS ^

function varargout = plot(varargin)

DESCRIPTION ^

```
PLOT plots a specwin object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PLOT plots a specwin object.

CALL:
      plot(specwin)
      h = plot(specwin)

M-FILE INFO: Get information about this methods by calling
              >> specwin.getInfo('plot')

              Get information about a specified set-plist by calling:
              >> specwin.getInfo('plot', 'set')

VERSION:     $Id: plot.m,v 1.9 2008/09/04 15:29:31 ingo Exp $

HISTORY:     06-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [plot](#) PLOT plots a specwin object.

This function is called by:

 [plot](#) PLOT plots a specwin object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % PLOT plots a specwin object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PLOT plots a specwin object.
0005 %
0006 % CALL:
0007 %       plot(specwin)
0008 %       h = plot(specwin)
0009 % M-FILE INFO: Get information about this methods by calling
0010 %              >> specwin.getInfo('plot')
0011 %
0012 %              Get information about a specified set-plist by calling:
0013 %              >> specwin.getInfo('plot', 'set')
```

Description of plot

```
0014 %
0015 % VERSION:      $Id: plot.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      06-02-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021 %
0022 function varargout = plot(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 % Get specwin objects
0031 ws = utils.helper.collect_objects(varargin(:), 'specwin');
0032
0033 hl = [];
0034
0035 hold on
0036 grid on;
0037
0038 colors = getappdata(0, 'ltpda_default_plot_colors');
0039
0040 titleStr = '';
0041 legendStr = '';
0042
0043 for i=1:numel(ws)
0044     w = ws(i);
0045     hl = [hl plot(w.win)];
0046     col = colors(mod(i-1, length(colors))+1);
0047     set(hl(end), 'Color', col);
0048     xlabel('sample');
0049     ylabel('amplitude');
0050     titleStr = [titleStr, utils.prog.label(w.type), ', '];
0051     lstr = [sprintf('alpha = %g\n', w.alpha)...
0052            sprintf('psll = %g\n', w.psll)...
0053            sprintf('rov = %g\n', w.rov)...
0054            sprintf('nenbw = %g\n', w.nenbw)...
0055            sprintf('w3db = %g\n', w.w3db)...
0056            sprintf('flatness = %g\n', w.flatness)];
0057     legendStr = [legendStr cellstr(lstr)];
0058 end
0059
0060 legend(legendStr);
0061 titleStr = titleStr(1:end-2);
0062 title(sprintf('Window: %s', titleStr));
0063
0064 if nargout > 0
0065     varargout{1} = hl;
0066 end
0067
0068 hold off
0069 end
0070
0071 %%%%%%%%%%%%%%
0072 %                                Local Functions                                %
0073 %%%%%%%%%%%%%%
0074 %
0075 %%%%%%%%%%%%%%
0076 %
0077 % FUNCTION:      getInfo
0078 %
0079 % DESCRIPTION:   Get Info Object
0080 %
0081 % HISTORY:      11-07-07 M Hewitson
0082 %                Creation.
0083 %
0084 %%%%%%%%%%%%%%
0085 %
0086 function ii = getInfo(varargin)
0087     if nargin == 1 && strcmpi(varargin{1}, 'None')
0088         sets = {};
0089         pl = [];
0090     else
0091         sets = {'Default'};
0092         pl = getDefaultPlist;
0093     end
0094     % Build info object
0095     ii = minfo(mfilename, 'specwin', '', utils.const.categories.output, '$Id: plot.m,v 1.9
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0096 end
0097
0098 %%%%%%%%%%%%%%
```

```
0099 %  
0100 % FUNCTION:      getDefaultPlist  
0101 %  
0102 % DESCRIPTION: Get Default Plist  
0103 %  
0104 % HISTORY:        11-07-07 M Hewitson  
0105 %                  Creation.  
0106 %  
0107 %%%%%%%%%%%%%%%  
0108 %  
0109 function plo = getDefaultPlist()  
0110     plo = plist();  
0111 end  
0112
```


string

PURPOSE ^

STRING writes a command string that can be used to recreate the input window object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input window object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate the
             input window object.

CALL:       cmd = string(sw)

The following call returns an minfo object describing this method:
>> info = specwin.getInfo('string')

VERSION:    $Id: string.m,v 1.8 2008/09/04 15:29:31 ingo Exp $

HISTORY:    29-03-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input window object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input window object.
0006 %
0007 % CALL:       cmd = string(sw)
0008 %
0009 % The following call returns an minfo object describing this method:
0010 %
0011 % >> info = specwin.getInfo('string')
0012 %
0013 % VERSION:    $Id: string.m,v 1.8 2008/09/04 15:29:31 ingo Exp $
0014 %
0015 % HISTORY:    29-03-2007 M Hewitson
0016 %               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 function varargout = string(varargin)
0021
```

```

0022 %%% Check if this is a call for parameters
0023 if utils.helper.isinfocall(varargin{:})
0024     varargout{1} = getInfo(varargin{3});
0025     return
0026 end
0027
0028 % Get specwin objects
0029 sw = utils.helper.collect_objects(varargin(:), 'specwin');
0030
0031 if length(sw) > 1
0032     cmd = '[';
0033 else
0034     cmd = '';
0035 end
0036
0037 for j=1:length(sw)
0038     switch sw(j).type
0039         case 'kaiser'
0040             cmd = [cmd ' specwin('' sw(j).type '', ' num2str(length(sw(j).win)) ', '
num2str(sw(j).psll) ')'];
0041         otherwise
0042             cmd = [cmd ' specwin('' sw(j).type '', ' num2str(length(sw(j).win)) ')'];
0043     end
0044 end
0045
0046 if length(sw) > 1
0047     cmd = [cmd ']';
0048 end
0049
0050 % Set output
0051 varargout{1} = cmd;
0052
0053 end
0054
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056 %                                     Local Functions                                     %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060 %
0061 % FUNCTION:      getInfo
0062 %
0063 % DESCRIPTION:   Get Info Object
0064 %
0065 % HISTORY:       11-07-07 M Hewitson
0066 %                 Creation.
0067 %
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069
0070 function ii = getInfo(varargin)
0071     if nargin == 1 && strcmpi(varargin{1}, 'None')
0072         sets = {};
0073         pl = [];
0074     else
0075         sets = {'Default'};
0076         pl = getDefaultPlist;
0077     end
0078     % Build info object
0079     ii = minfo(mfilename, 'specwin', '', utils.const.categories.output, '$Id: string.m,v
1.8 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0080 end
0081
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083 %
0084 % FUNCTION:      getDefaultPlist
0085 %
0086 % DESCRIPTION:   Get Default Plist
0087 %
0088 % HISTORY:       11-07-07 M Hewitson
0089 %                 Creation.
0090 %
0091 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0092
0093 function plo = getDefaultPlist()
0094     plo = plist();
0095 end
0096

```

minus

PURPOSE ^

MINUS overloads - operator for time objects.

SYNOPSIS ^

function varargout = minus(varargin)

DESCRIPTION ^

```
MINUS overloads - operator for time objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MINUS overloads - operator for time objects.
             This function subtracts a time in seconds from a time object.
             It is possible to define the subtrahend as a string,
             a vector of numbers, an ao containing a vector of numbers,
             or a time object.

CALL:       t2 = t1 - value (in seconds)
             sec = t1 - t2;

EXAMPLES:   >> t1      = time('2007-07-01 12:23:33');
             >> t2      = t1 - [30 10];
             >> t2      = t1 - '30';
             >> t2      = t1 - ao(plist('vals',[30 10]));
             >> seconds = t1 - t2;

M-FILE INFO: Get information about this methods by calling
             >> time.getInfo('minus')

             Get information about a specified set-plist by calling:
             >> time.getInfo('minus','set')

VERSION:    $Id: minus.m,v 1.12 2008/09/04 15:29:31 ingo Exp $

HISTORY:    03-08-2007 Diepholz
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % MINUS overloads - operator for time objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: MINUS overloads - operator for time objects.
0005 % This function subtracts a time in seconds from a time object.
0006 % It is possible to define the subtrahend as a string,
0007 % a vector of numbers, an ao containing a vector of numbers,
0008 % or a time object.
0009 %
0010 % CALL:      t2 = t1 - value (in seconds)
0011 %           sec = t1 - t2;
0012 %
0013 % EXAMPLES:  >> t1      = time('2007-07-01 12:23:33');
0014 %           >> t2      = t1 - [30 10];
0015 %           >> t2      = t1 - '30';
0016 %           >> t2      = t1 - ao(plist('vals',[30 10]));
0017 %           >> seconds = t1 - t2;
0018 %
0019 % M-FILE INFO: Get information about this methods by calling
0020 %           >> time.getInfo('minus')
0021 %
0022 %           Get information about a specified set-plist by calling:
0023 %           >> time.getInfo('minus', 'set')
0024 %
0025 % VERSION:    $Id: minus.m,v 1.12 2008/09/04 15:29:31 ingo Exp $
0026 %
0027 % HISTORY:    03-08-2007 Diepholz
0028 %              Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031 %
0032 function varargout = minus(varargin)
0033 %
0034 %%% Check if this is a call for parameters
0035 if utils.helper.isinfocall(varargin{:})
0036     varargout{1} = getInfo(varargin{3});
0037     return
0038 end
0039 %
0040 % Collects the input values
0041 t1 = varargin{1};
0042 t2 = varargin{2};
0043 %
0044 %%% decide whether we modify the time-object, or create a new one.
0045 t1 = copy(t1, nargout);
0046 %
0047 %%%>> t2 = t1 - ao(30); %%%
0048 if isa(t2, 'ao')
0049     t2 = t2.data.y;
0050 end
0051 %
0052 %%%>> t2 = t1 - '30'; %%%
0053 if ischar(t2)
0054     t2 = str2double(t2);
0055 end
0056 %
0057 %%%>> t2 = t1 - 30; %%%
0058 if isnumeric(t2)
0059 %
0060 % Loop over the list of inputs
0061 for jj = 1 : length(t1)
0062     t1(jj).setEpochtime(t1(jj).utc_epoch_milli - t2*1000, 'internal');
0063 end
0064 res = t1;
0065 %
0066 %%%>> number = t1 - t2; %%%
0067 elseif isa(t2, 'time')
0068     res = (t1.utc_epoch_milli - t2.utc_epoch_milli)/1000;
0069 else
0070     error('### Unknown minus-operation to a time object. ');
0071 end
0072 %
0073 varargout{1} = res;
0074 %
0075 end
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 % Local Functions %
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080 %
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %
0083 % FUNCTION:      getInfo
0084 %
0085 % DESCRIPTION:   Get Info Object
0086 %
0087 % HISTORY:       11-07-07 M Hewitson
0088 %                 Creation.
0089 %

```

```
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091
0092 function ii = getInfo(varargin)
0093     if nargin == 1 && strcmpi(varargin{1}, 'None')
0094         sets = {};
0095         pl = [];
0096     else
0097         sets = {'Default'};
0098         pl = getDefaultPlist;
0099     end
0100     % Build info object
0101     ii = minfo(mfilename, 'time', '', utils.const.categories.aop, '$Id: minus.m,v 1.12
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0102 end
0103
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %
0106 % FUNCTION:      getDefaultPlist
0107 %
0108 % DESCRIPTION:   Get Default Plist
0109 %
0110 % HISTORY:       11-07-07 M Hewitson
0111 %                 Creation.
0112 %
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114
0115 function plo = getDefaultPlist()
0116     plo = plist();
0117 end
0118
```

plus

PURPOSE ^

PLUS overloads + operator for time objects.

SYNOPSIS ^

function varargout = plus(varargin)

DESCRIPTION ^

```
PLUS overloads + operator for time objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PLUS overloads + operator for time objects.
             This function adds a time in seconds to a time object.
             It is possible to define the addend as a string,
             a vector of numbers, an ao containing a vector of numbers,
             or a time object.

CALL:       t2 = t1 + value (in seconds)
             sec = t1 + t2;

EXAMPLES:   >> t1      = time('2007-07-01 12:23:33');
             >> t2      = t1 + [30 2330];
             >> t2      = t1 + '30';
             >> t2      = t1 + ao(plist('vals',[30 2330]));
             >> seconds = t1 + t2;

M-FILE INFO: Get information about this methods by calling
             >> time.getInfo('plus')

             Get information about a specified set-plist by calling:
             >> time.getInfo('plus', 'set')

VERSION:    $Id: plus.m,v 1.15 2008/09/04 15:29:31 ingo Exp $

HISTORY:    03-08-2007 Diepholz
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % PLUS overloads + operator for time objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: PLUS overloads + operator for time objects.
0005 % This function adds a time in seconds to a time object.
0006 % It is possible to define the addend as a string,
0007 % a vector of numbers, an ao containing a vector of numbers,
0008 % or a time object.
0009 %
0010 % CALL:      t2 = t1 + value (in seconds)
0011 %           sec = t1 + t2;
0012 %
0013 % EXAMPLES:  >> t1      = time('2007-07-01 12:23:33');
0014 %           >> t2      = t1 + [30 2330];
0015 %           >> t2      = t1 + '30';
0016 %           >> t2      = t1 + ao(plist('vals',[30 2330]));
0017 %           >> seconds = t1 + t2;
0018 %
0019 % M-FILE INFO: Get information about this methods by calling
0020 %           >> time.getInfo('plus')
0021 %
0022 %           Get information about a specified set-plist by calling:
0023 %           >> time.getInfo('plus', 'set')
0024 %
0025 % VERSION:    $Id: plus.m,v 1.15 2008/09/04 15:29:31 ingo Exp $
0026 %
0027 % HISTORY:    03-08-2007 Diepholz
0028 %             Creation
0029 %
0030 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0031 %
0032 function varargout = plus(varargin)
0033 %
0034 %%% Check if this is a call for parameters
0035 if utils.helper.isinfocall(varargin{:})
0036     varargout{1} = getInfo(varargin{3});
0037     return
0038 end
0039 %
0040 %%% Collects the input values
0041 t1 = varargin{1};
0042 t2 = varargin{2};
0043 %
0044 %%% decide whether we modify the time-object, or create a new one.
0045 t1 = copy(t1, nargout);
0046 %
0047 %%%>> t2 = t1 + ao(30); %%%
0048 if isa(t2, 'ao')
0049     t2 = t2.data.y;
0050 end
0051 %
0052 %%%>> t2 = t1 + '30'; %%%
0053 if ischar(t2)
0054     t2 = str2double(t2);
0055 end
0056 %
0057 %%%>> t2 = t1 + 30; %%%
0058 if isnumeric(t2)
0059 %
0060 % Loop over the list of inputs
0061 for jj = 1 : length(t1)
0062     t1(jj).setEpochtime(t1(jj).utc_epoch_milli + t2*1000, 'internal');
0063 end
0064 res = t1;
0065 %
0066 %%%>> number = t1 + t2; %%%
0067 elseif isa(t2, 'time')
0068     res = t1.utc_epoch_milli + t2.utc_epoch_milli;
0069 else
0070     error('### Unknown plus-operation to a time object.');
```

```

0071 end
0072
0073 varargout{1} = res;
0074 end
0075 %
0076 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077 % Local Functions %
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 % FUNCTION:      getInfo
0083 %
0084 % DESCRIPTION:  Get Info Object
0085 %
0086 % HISTORY:      11-07-07 M Hewitson
0087 %               Creation.
0088 %
0089 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
0090
0091 function ii = getInfo(varargin)
0092     if nargin == 1 && strcmpi(varargin{1}, 'None')
0093         sets = {};
0094         pl = [];
0095     else
0096         sets = {'Default'};
0097         pl = getDefaultPlist;
0098     end
0099     % Build info object
0100     ii = minfo(mfilename, 'time', '', utils.const.categories.aop, '$Id: plus.m,v 1.15
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0101 end
0102
0103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104 %
0105 % FUNCTION:      getDefaultPlist
0106 %
0107 % DESCRIPTION:  Get Default Plist
0108 %
0109 % HISTORY:      11-07-07 M Hewitson
0110 %                Creation.
0111 %
0112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113
0114 function plo = getDefaultPlist()
0115     plo = plist();
0116 end
0117
```


time

PURPOSE ^

TIME time object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
TIME time object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION:  TIME time object class constructor.
              Create a time object.

SUPER CLASSES: ltpda_nuo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
        version          - cvs-version string.

    Protected Properties (read only)
        utc_epoch_milli  - Unix epoch time of the utc time.
        timezone         - Timezone of the date/time
        timeformat       - Matlab timeformat of the date/time
        time_str         - Time string of the unix time in the specified
                          timezone.

TIME METHODS:

    Defined Abstract methods:
        char             - returns one character string which represents the object
        copy             - copies an object
        display          - displays an object
        update_struct    - updates a object structure to the current tbx-version

    Public methods:
        format           - returns a time-string in specified format.
        getTimezones     - get all possible timezones
        minus            - overloads '-' operator for time objects.
        plus             - overloads '+' operator for time objects.
        setEpochtime    - set the 'utc_epoch_milli' property
        setTimeformat    - set the 'timeformat' property
        setTime_str      - set the 'time_str' property
        setTimezone      - set the 'timezone' property
        string           - writes a command string that can be used to recreate the object

CONSTRUCTORS:

    t1 = time()
    t1 = time('14:00:05.000')
    t1 = time('2007-06-06 14:00:05')
    t1 = time('14:00:05 2007-06-06')
    t1 = time(1186135200000)
    t1 = time(plist)
    t1 = time(1186135200000, 'HH:MM:SS')

Parameter sets for plist constructor (in order of priority):

From XML File
-----

    Construct an TIME by loading it from an XML file.

    'filename' - construct an TIME from a filename.
                Example: plist('filename', 't1.xml')
                [default: empty string]

From MAT File
```

```

-----
Construct an TIME by loading it from a MAT file.

'filename' - construct an TIME from a filename.
             Example: plist('filename', 't1.mat')
             [default: empty string]

From Repository
-----

Construct an TIME by retrieving it from an LTPDA repository.

'Hostname' - the repository hostname. Only those objects which
             are TIMES are returned.
             [default: 'localhost'];

             Additional parameters:

             'Database' - The database name [default: 'ltpda']
             'ID'       - A vector of object IDs. [default: []]

From Time Definition
-----
Construct an TIME by its properties definition
'utc_epoch_milli' - the time in milliseconds [default: 0];

             Additional parameters:

             'timezone' - Timezone (string or java object) [default: 'UTC']
             'timeformat' - Time format (string) [default: 'yyyy-mm-dd HH:MM:SS.FFF']%

From Time String
-----
Construct an TIME by its properties definition
'time_str' - the time string [default: '1970-01-01 00:00:00.000'];

             Additional parameters:

             'timezone' - Timezone (string or java object) [default: 'UTC']
             'timeformat' - Time format (string) [default: 'yyyy-mm-dd HH:MM:SS.FFF']

From Plist
-----

'Plist' - construct from a plist. The value passed should be a plist
         object.
         [default: empty plist]

M-FILE INFO: The following call returns an minfo object that contains
              information about the time constructor:
              >> info = time.getInfo
              or  >> info = time.getInfo('time')

              You can get information about class methods by calling:
              >> info = time.getInfo(method)
              e.g. >> info = time.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
              >> info = time.getInfo(method, set)
              e.g. >> info = time.getInfo('time', 'From Time String')

VERSION:      $Id: time.m,v 1.56 2008/09/03 16:39:15 hewitson Exp $

HISTORY:      23-07-2007 Diepholz
               Creation

```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY Make copy of time objects depending of the second input
-  **epoch2str** EPOCH2STR Returns computes the time sting of the epoch time stored in 'utc_epoch_milli' and 'created'.
-  **format** FORMAT Returns the time in specified format.

-  **fromTimeString** FROMTimestRING creates a time object from a time string.
-  **fromTimeValue** FROMTIMEVALUE creates a time object from unix epoch time.
-  **getTimezones** GETTIMEZONES Get all possible timezones.
-  **minus** MINUS overloads - operator for time objects.
-  **parse_time_string** PRASE_TIME_STRING parses the time string.
-  **plus** PLUS overloads + operator for time objects.
-  **setEpochtime** SETEPOCHTIME Set the property 'utc_epoch_milli'.
-  **setTime_str** SETTIME_STR Set the property 'time_str'.
-  **setTimeformat** SETTIMEFORMAT Set the property 'timeformat'.
-  **setTimezone** SETTIMEZONE Set the property 'timezone'.
-  **string** STRING writes a command string that can be used to recreate the input time object.
-  **time** TIME time object class constructor.
-  **update_struct** UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  **copy** COPY Make copy of time objects depending of the second input
-  **time** TIME time object class constructor.

SUBFUNCTIONS ^

-  **function** obj = time(varargin)
-  **function** ii = getInfo(varargin)
-  **function** out = VEROUT()
-  **function** out = SETS()
-  **function** out = getDefaultPlist(set)

SOURCE CODE ^

```

0001 % TIME time object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION:  TIME time object class constructor.
0005 %               Create a time object.
0006 %
0007 % SUPER CLASSES: ltpda_nuo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         version          - cvs-version string.
0013 %
0014 %     Protected Properties (read only)
0015 %         utc_epoch_milli   - Unix epoch time of the utc time.
0016 %         timezone          - Timezone of the date/time
0017 %         timeformat        - Matlab timeformat of the date/time
0018 %         time_str          - Time string of the unix time in the specified
0019 %                             timezone.
0020 %
0021 % TIME METHODS:
0022 %
0023 %     Defined Abstract methods:
0024 %         char              - returns one character string which represents the object
0025 %         copy              - copies an object
0026 %         display           - displays an object
0027 %         update_struct     - updates a object structure to the current tbx-version
0028 %
0029 %     Public methods:
0030 %         format            - returns a time-string in specified format.
0031 %         getTimezones      - get all possible timezones
0032 %         minus             - overloads '-' operator for time objects.
0033 %         plus              - overloads '+' operator for time objects.
0034 %         setEpochtime     - set the 'utc_epoch_milli' property
0035 %         setTimeformat     - set the 'timeformat' property
0036 %         setTime_str       - set the 'time_str' property
0037 %         setTimezone       - set the 'timezone' property
0038 %         string            - writes a command string that can be used to recreate the object

```

```

0039 %
0040 % CONSTRUCTORS:
0041 %
0042 %         t1 = time()
0043 %         t1 = time('14:00:05.000')
0044 %         t1 = time('2007-06-06 14:00:05')
0045 %         t1 = time('14:00:05 2007-06-06')
0046 %         t1 = time(1186135200000)
0047 %         t1 = time(plist)
0048 %         t1 = time(1186135200000, 'HH:MM:SS')
0049 %
0050 % Parameter sets for plist constructor (in order of priority):
0051 %
0052 % From XML File
0053 % -----
0054 %
0055 %     Construct an TIME by loading it from an XML file.
0056 %
0057 %     'filename' - construct an TIME from a filename.
0058 %                  Example: plist('filename', 't1.xml')
0059 %                  [default: empty string]
0060 %
0061 %
0062 % From MAT File
0063 % -----
0064 %
0065 %     Construct an TIME by loading it from a MAT file.
0066 %
0067 %     'filename' - construct an TIME from a filename.
0068 %                  Example: plist('filename', 't1.mat')
0069 %                  [default: empty string]
0070 %
0071 %
0072 % From Repository
0073 % -----
0074 %
0075 %     Construct an TIME by retrieving it from an LTPDA repository.
0076 %
0077 %     'Hostname' - the repository hostname. Only those objects which
0078 %                  are TIMES are returned.
0079 %                  [default: 'localhost'];
0080 %
0081 %     Additional parameters:
0082 %
0083 %         'Database' - The database name [default: 'ltpda']
0084 %         'ID'       - A vector of object IDs. [default: []]
0085 %
0086 %
0087 % From Time Definition
0088 % -----
0089 %     Construct an TIME by its properties definition
0090 %     'utc_epoch_milli' - the time in milliseconds [default: 0];
0091 %
0092 %     Additional parameters:
0093 %
0094 %         'timezone' - Timezone (string or java object) [default: 'UTC']
0095 %         'timeformat' - Time format (string) [default: 'yyyy-mm-dd
HH:MM:SS.FFF']%
0096 %
0097 % From Time String
0098 % -----
0099 %     Construct an TIME by its properties definition
0100 %     'time_str'      - the time string [default: '1970-01-01 00:00:00.000'];
0101 %
0102 %     Additional parameters:
0103 %
0104 %         'timezone' - Timezone (string or java object) [default: 'UTC']
0105 %         'timeformat' - Time format (string) [default: 'yyyy-mm-dd
HH:MM:SS.FFF']
0106 %
0107 %
0108 % From Plist
0109 % -----
0110 %
0111 %     'Plist' - construct from a plist. The value passed should be a plist
0112 %                object.
0113 %                [default: empty plist]
0114 %
0115 % M-FILE INFO: The following call returns an minfo object that contains
0116 %                information about the time constructor:
0117 %                >> info = time.getInfo
0118 %                or  >> info = time.getInfo('time')
0119 %
0120 %                You can get information about class methods by calling:
0121 %                >> info = time.getInfo(method)
0122 %                e.g. >> info = time.getInfo('eq')

```

```

0123 %
0124 %
0125 %         You can also restrict the sets of parameters contained in
0126 %         the minfo object by calling:
0127 %             >> info = time.getInfo(method, set)
0128 %         e.g. >> info = time.getInfo('time', 'From Time String')
0129 %
0130 %
0131 % VERSION:      $Id: time.m,v 1.56 2008/09/03 16:39:15 hewitson Exp $
0132 %
0133 %
0134 % HISTORY:      23-07-2007 Diepholz
0135 %                Creation
0136 %
0137 %-----
0138 %----- Private read-only Properties -----
0139 %-----
0140 properties (SetAccess = private)
0141     utc_epoch_milli = 0;
0142     timezone        = '';
0143     timeformat      = '';
0144     time_str        = '';
0145 end
0146 %----- Private Properties -----
0147 properties (GetAccess = protected, SetAccess = protected)
0148 end
0149
0150 %----- Protected Properties -----
0151 properties (SetAccess = protected)
0152     version          = '$Id: time.m,v 1.56 2008/09/03 16:39:15 hewitson Exp $';
0153 end
0154
0155 %%%%%%%%%%%
0156 %                Check property setting
0157 %%%%%%%%%%%
0158
0159 %%%%%%%%%%%
0160 %                Constructor
0161 %%%%%%%%%%%
0162
0163 methods
0164     function obj = time(varargin)
0165
0166         %%% Call superclass
0167         obj = obj@ltpda_nuo(varargin{:});
0168
0169         %%%%%%%%%%% Set default values %%%%%%%%%%%
0170         %%% Except for a struct as an input
0171         if ~(nargin == 1 && isstruct(varargin{1}))
0172             import java.util.GregorianCalendar;
0173             import java.util.TimeZone;
0174             unix_time = getTime(java.util.Date);
0175             obj.timezone = java.util.TimeZone.getTimeZone(getappdata(0, 'timezone'));
0176             obj.timeformat = getappdata(0, 'time_format_str');
0177             % Define local/utc timezone
0178             obj.timezone = TimeZone.getTimeZone(getappdata(0, 'timezone'));
0179         end
0180
0181         switch nargin
0182         case 0
0183             %%%%%%%%%%% no input %%%%%%%%%%%
0184             %%%%%%%%%%% t1 = time() %%%%%%%%%%%
0185             obj.utc_epoch_milli = unix_time;
0186         case 1
0187             %%%%%%%%%%% one input %%%%%%%%%%%
0188             if isa(varargin{1}, 'time')
0189                 %%%%%%%%%%% t1 = time(time-object) %%%%%%%%%%%
0190                 obj = copy(varargin{1}, 1);
0191             elseif ischar(varargin{1})
0192                 %%%%%%%%%%% t1 = time('foo.xml') %%%%%%%%%%%
0193                 %%%%%%%%%%% t1 = time('foo.mat') %%%%%%%%%%%
0194                 [path, name, ext] = fileparts(varargin{1});
0195                 if ~isempty(ext) && length(ext) == 4 && isempty(str2num(ext(2)))
0196                     error('### It is not allowed to read a time-object');
0197                 else
0198                     %%%%%%%%%%% t1 = time('2007-08-03 10:00:00') %%%%%%%%%%%
0199                     obj.setTime_str(varargin{1}, 'internal');
0200                 end
0201             end
0202         end
0203

```

```

0209 elseif isstruct(varargin{1})
0210     %%%%%%%%%%% t1 = time(structure) %%%%%%%%%%%
0211
0212     %% Set properties which are declared in this class
0213     t_struct = varargin{1};
0214
0215     obj.utc_epoch_milli = t_struct.utc_epoch_milli;
0216     obj.timezone        = t_struct.timezone;
0217     obj.timeformat      = t_struct.timeformat;
0218     obj.time_str        = t_struct.time_str;
0219     obj.version         = t_struct.version;
0220
0221 elseif isnumeric(varargin{1})
0222
0223     %%%%%%%%%%% t1 = time(1214914600000) %%%%%%%%%%%
0224     % We utc epoch milliseconds as input
0225     obj.utc_epoch_milli = varargin{1};
0226
0227 elseif isa(varargin{1}, 'plist')
0228
0229     %%%%%%%%%%% t1 = time(plist) %%%%%%%%%%%
0230     pl = varargin{1};
0231     hostname = find(pl, 'hostname');
0232     pl_epoch_milli = find(pl, 'utc_epoch_milli');
0233     pl_time_str = find(pl, 'time_str');
0234     ipl = find(pl, 'plist');
0235
0236     if ~isempty(hostname)
0237         %-----
0238         %--- Construct from repository
0239         %-----
0240         % do hostname constructor
0241         obj = fromRepository(obj, pl, VERSION, ALGONAME);
0242
0243     elseif ~isempty(pl_epoch_milli)
0244         %-----
0245         %--- Construct from millisecs time definition
0246         %-----
0247         obj = fromTimeValue(obj, pl);
0248
0249     elseif ~isempty(pl_time_str)
0250         %-----
0251         %--- Construct from string time definition
0252         %-----
0253         obj = fromTimeString(obj, pl);
0254
0255     elseif ~isempty(ipl)
0256         %-----
0257         %--- Construct from plist
0258         %-----
0259
0260         % if the plist is empty, we return the default time object
0261         if nparams(ipl) == 0
0262             % default time object
0263         else
0264             % do plist constructor
0265             obj = time(ipl);
0266         end
0267
0268     else
0269         %-----
0270         %--- ERROR
0271         %-----
0272
0273         %% if the plist is empty then return the default time object
0274         if nparams(pl) == 0
0275             % default time object
0276         else
0277             error('### Unknown TIME constructor method.');

```



```

0295         elseif ischar(varargin{1}) && ischar(varargin{2})
0296             %%%%%%%%%%% t1 = time('14:00:00', 'HH:MM:SS') %%%%%%%%%%%
0297             obj.setTime_str(varargin{1});
0298             obj.setTimeformat(varargin{2});
0299
0300         else
0301             error('### Unknown constructor with two inputs.');
```

0302 end

0303

0304 otherwise

0305 error('### Unknown number of constructor arguments.');

0306 end % End switch over input args

0307 end % End time()

0308 end % End constructor

0309

0310 %%%%%%%%%%%

0311 % Methods (Public) %

0312 %%%%%%%%%%%

0313

0314 methods

0315 varargout = copy(varargin)

0316 varargout = format(varargin)

0317 varargout = getTimezones(varargin)

0318 varargout = minus(varargin)

0319 varargout = plus(varargin)

0320 varargout = string(varargin)

0321

0322 varargout = setEpochtime(varargin)

0323 varargout = setTimezone(varargin)

0324 varargout = setTimeformat(varargin)

0325 varargout = setTime_str(varargin)

0326 end % End public methods

0327

0328 %%%%%%%%%%%

0329 % Methods (protected) %

0330 %%%%%%%%%%%

0331

0332

0333 methods (Access = protected)

0334 end

0335

0336 %%%%%%%%%%%

0337 % Methods (private) %

0338 %%%%%%%%%%%

0339

0340 methods (Access = private)

0341 varargout = fromTimeValue(varargin)

0342 varargout = fromTimeString(varargin)

0343 end

0344 %%%%%%%%%%%

0345 % Methods (static) %

0346 %%%%%%%%%%%

0347 methods (Static)

0348

0349 varargout = update_struct(varargin);

0350

0351 function ii = getInfo(varargin)

0352 ii = utils.helper.generic_getInfo(varargin{:}, 'time');

0353 end

0354

0355 function out = VEROUT()

0356 out = '\$Id: time.m,v 1.56 2008/09/03 16:39:15 hewitson Exp \$';

0357 end

0358

0359 function out = SETS()

0360 out = {...

0361 'Default', ...

0362 'From Repository', ...

0363 'From Time Definition', ...

0364 'From Time String', ...

0365 'From Plist'};

0366 end

0367

0368 function out = getDefaultPlist(set)

0369

0370 switch set

0371 case 'Default'

0372 out = plist();

0373 case 'From Repository'

0374 %--- Repository constructor

0375 out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);

0376 case 'From Time Definition'

0377 %--- Definition constructor

0378 out = plist(...

0379 'utc_epoch_milli', 0, ...

0380 'timezone', 'UTC', ...

```
0381         'timeformat', 'yyyy-mm-dd HH:MM:SS.FFF');
0382     case 'From Time String'
0383         %--- String constructor
0384         out = plist(...
0385             'time_str', '1970-01-01 00:00:00.000', ...
0386             'timezone', 'UTC', ...
0387             'timeformat', 'yyyy-mm-dd HH:MM:SS.FFF');
0388         %--- Create from a plist
0389     case 'From Plist'
0390         out = plist('Plist', []);
0391     otherwise
0392         error('### Unknown set [%s] to get the default plist.', set);
0393     end
0394 end % End getDefaultPlist
0395
0396 end % End static methods
0397
0398 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0399 %                                     Methods (static, private) %
0400 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0401
0402 methods (Static, Access=private)
0403
0404     varargout = parse_time_string(varargin)
0405     [time_str, created_str] = epoch2str(tt)
0406
0407 end % End static, private methods
0408 end % End clasdef
0409
0410
```


getTimezones

PURPOSE ^

GETTIMEZONES Get all possible timezones.

SYNOPSIS ^

function varargout = getTimezones(varargin)

DESCRIPTION ^

```
GETTIMEZONES Get all possible timezones.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get all possible timezones.

CALL:      zones = obj.getTimezones();

INPUTS:    obj - can be a vector, matrix, list, or a mix of them.

M-FILE INFO: Get information about this methods by calling
              >> time.getInfo('getTimezones')

              Get information about a specified set-plist by calling:
              >> time.getInfo('getTimezones', 'set')

VERSION:    $Id: getTimezones.m,v 1.2 2008/09/04 15:29:31 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETTIMEZONES Get all possible timezones.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get all possible timezones.
0005 %
0006 % CALL:      zones = obj.getTimezones();
0007 %
0008 % INPUTS:    obj - can be a vector, matrix, list, or a mix of them.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %              >> time.getInfo('getTimezones')
0012 %
0013 %              Get information about a specified set-plist by calling:
0014 %              >> time.getInfo('getTimezones', 'set')
0015 %
```

```
0016 % VERSION:      $Id: getTimezones.m,v 1.2 2008/09/04 15:29:31 ingo Exp $
0017 %
0018 % HISTORY:       27-05-2008 Diepholz
0019 %               Creation
0020 %
0021 %%%%%%%%%%%%%%
0022
0023 function varargout = getTimezones(varargin)
0024
0025     %% Check if this is a call for parameters
0026     if utils.helper.isinfocall(varargin{:})
0027         varargout{1} = getInfo(varargin{3});
0028         return
0029     end
0030
0031     varargout{1} = cell(java.util.TimeZone.getAvailableIDs);
0032
0033 end
0034
0035 %%%%%%%%%%%%%%
0036 %               Local Functions
0037 %%%%%%%%%%%%%%
0038
0039 %%%%%%%%%%%%%%
0040 %
0041 % FUNCTION:      getInfo
0042 %
0043 % DESCRIPTION:   Get Info Object
0044 %
0045 % HISTORY:       11-07-07 M Hewitson
0046 %               Creation.
0047 %
0048 %%%%%%%%%%%%%%
0049
0050 function ii = getInfo(varargin)
0051     if nargin == 1 && strcmpi(varargin{1}, 'None')
0052         sets = {};
0053         pl   = [];
0054     else
0055         sets = {'Default'};
0056         pl   = getDefaultPlist;
0057     end
0058     % Build info object
0059     ii = minfo(mfilename, 'time', '', utils.const.categories.helper, '$Id:
getTimezones.m,v 1.2 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0060 end
0061
0062 %%%%%%%%%%%%%%
0063 %
0064 % FUNCTION:      getDefaultPlist
0065 %
0066 % DESCRIPTION:   Get Default Plist
0067 %
0068 % HISTORY:       11-07-07 M Hewitson
0069 %               Creation.
0070 %
0071 %%%%%%%%%%%%%%
0072
0073 function plo = getDefaultPlist()
0074     plo = plist();
0075 end
0076
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input time object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input time object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: STRING writes a command string that can be used to recreate the
             input time object.
CALL:       cmd = string(time-object);
M-FILE INFO: Get information about this methods by calling
              >> time.getInfo('string')
              Get information about a specified set-plist by calling:
              >> time.getInfo('string', 'set')
VERSION:    $Id: string.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
HISTORY:    07-08-2007 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a time object into a string.
-  [string](#) STRING writes a command string that can be used to recreate the input time object.

This function is called by:

-  [string](#) STRING writes a command string that can be used to recreate the input time object.
-  [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input time object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input time object.
0006 %
0007 % CALL:       cmd = string(time-object);
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
```

```

0010 %                >> time.getInfo('string')
0011 %
0012 %                Get information about a specified set-plist by calling:
0013 %                >> time.getInfo('string', 'set')
0014 %
0015 % VERSION:      $Id: string.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      07-08-2007 Diepholz
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021 %
0022 function varargout = string(varargin)
0023 %
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029 %
0030 % collect the time-objects
0031 tt = utils.helper.collect_objects(varargin(:), 'time');
0032 %
0033 cmd = '';
0034 for jj=1:length(tt)
0035     pl = plist();
0036     pl.append('timeformat', char(tt(jj).timeformat));
0037     pl.append('timezone', char(tt(jj).timezone.getID));
0038     pl.append('utc_epoch_milli', tt(jj).utc_epoch_milli);
0039 %
0040     cmd = [cmd 'time(' string(pl) ') '];
0041 %
0042 end
0043 %
0044 %%% Wrap the command only with brackets if there are more than one object
0045 if numel(tt) > 1
0046     cmd = '[' cmd(1:end-1) '];'
0047 %
0048 end
0049 %
0050 % prepare output
0051 varargout{1} = cmd;
0052 %
0053 end
0054 %
0055 %%%%%%%%%%%%%%
0056 %                Local Functions
0057 %%%%%%%%%%%%%%
0058 %
0059 %%%%%%%%%%%%%%
0060 %
0061 % FUNCTION:      getInfo
0062 %
0063 % DESCRIPTION:   Get Info Object
0064 %
0065 % HISTORY:      11-07-07 M Hewitson
0066 %                Creation.
0067 %
0068 %%%%%%%%%%%%%%
0069 %
0070 function ii = getInfo(varargin)
0071 if nargin == 1 && strcmpi(varargin{1}, 'None')
0072     sets = {};
0073     pl = [];
0074 else
0075     sets = {'Default'};
0076     pl = getDefaultPlist;
0077 end
0078 % Build info object
0079 ii = minfo(mfilename, 'time', '', utils.const.categories.helper, '$Id: string.m,v 1.9
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0080 end
0081 %
0082 %%%%%%%%%%%%%%
0083 %
0084 % FUNCTION:      getDefaultPlist
0085 %
0086 % DESCRIPTION:   Get Default Plist
0087 %
0088 % HISTORY:      11-07-07 M Hewitson
0089 %                Creation.
0090 %
0091 %%%%%%%%%%%%%%
0092 %
0093 function plo = getDefaultPlist()
0094 plo = plist();

```

0095 end

0096

copy

PURPOSE ^

COPY Make copy of time objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

COPY Make copy of time objects depending of the second input
%%%

DESCRIPTION: COPY Make copy of time objects depending of the second input

CALL: objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
>> time.getInfo('copy')

VERSION: \$Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp \$

HISTORY: 11-07-2008 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

 [time](#) TIME time object class constructor.

This function is called by:

-  [fromTimeString](#) FROMTIMESTRING creates a time object from a time string.
-  [fromTimeValue](#) FROMTIMEVALUE creates a time object from unix epoch time.
-  [minus](#) MINUS overloads - operator for time objects.
-  [plus](#) PLUS overloads + operator for time objects.
-  [setEpochtime](#) SETEPOCHTIME Set the property 'utc_epoch_milli'.
-  [setTime_str](#) SETTIME_STR Set the property 'time_str'.
-  [setTimeformat](#) SETTIMEFORMAT Set the property 'timeformat'.
-  [setTimezone](#) SETTIMEZONE Set the property 'timezone'.
-  [time](#) TIME time object class constructor.

SOURCE CODE ^

```
0001 % COPY Make copy of time objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of time objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> time.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
```

```
0013 % HISTORY:      11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('time');
0022         s = size(old);
0023         obj(s(1),s(2)) = time;
0024         for kk=1:numel(old)
0025             utils.helper.msg(utils.const.msg.OPROC2, 'copying time');
0026             for jj=1:length(ps)
0027                 obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028             end
0029             % reset on copy
0030             obj(kk).version = time.VEROUT;
0031         end
0032     else
0033         obj = old;
0034     end
0035     varargout{1} = obj;
0036 end
0037
```

setEpochtime

PURPOSE ^

SETEPOCHTIME Set the property 'utc_epoch_milli'.

SYNOPSIS ^

function varargout = setEpochtime(varargin)

DESCRIPTION ^

SETEPOCHTIME Set the property 'utc_epoch_milli'.
%%%

DESCRIPTION: Set the property 'utc_epoch_milli'.

CALL:
obj = obj.setEpochtime(123456);
obj = obj.setEpochtime(plist('epochtime', 123456));
obj = setEpochtime(obj, 123456);

INPUTS:
obj - can be a vector, matrix, list, or a mix of them.
pl - to set the epochtime with a plist specify only one plist with
only one key-word 'epochtime'.

M-FILE INFO: Get information about this methods by calling
>> time.getInfo('setEpochtime')

Get information about a specified set-plist by calling:
>> time.getInfo('setEpochtime', 'set')

VERSION: \$Id: setEpochtime.m,v 1.5 2008/09/04 15:29:31 ingo Exp \$

HISTORY: 27-05-2008 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETEPOCHTIME Set the property 'utc_epoch_milli'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'utc_epoch_milli'.
0005 %
0006 % CALL:      obj = obj.setEpochtime(123456);
0007 %           obj = obj.setEpochtime(plist('epochtime', 123456));
0008 %           obj = setEpochtime(obj, 123456);
```



```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl  - to set the epochtime with a plist specify only one plist with
0012 %                   only one key-word 'epochtime'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> time.getInfo('setEpochtime')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> time.getInfo('setEpochtime', 'set')
0019 %
0020 % VERSION:     $Id: setEpochtime.m,v 1.5 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026
0027 function varargout = setEpochtime(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037
0038     %%% decide whether we modify the time-object, or create a new one.
0039     varargin{1} = copy(varargin{1}, nargout);
0040
0041     varargin{1}.utc_epoch_milli = varargin{2};
0042     varargout{1} = varargin{1};
0043     return
0044 end
0045
0046 %%% Normal call:
0047 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'time');
0048 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0049
0050 epochtime = rest;
0051
0052 if length(epochtime) ~=1
0053
0054     if length(pls) == 1
0055         epochtime = {};
0056         if nparams(pls) == 1 && isparam(pls, 'epochtime')
0057
0058             %%% If the plist contains only one parameter with the key-word 'epochtime'
0059             %%% then set 'epochtime' to the value in the time object.
0060             epochtime{1} = find(pls, 'epochtime');
0061
0062         else
0063             epochtime{1} = pls;
0064         end
0065     elseif length(pls) > 1
0066         error('### To set the ''epochtime'' please specify only one plist')
0067     else
0068         error('### Please specify [only one] value.')
0069     end
0070 end
0071
0072 %%% Set the epochtime
0073 for ii = 1:numel(objs)
0074     %%% decide whether we modify the time-object, or create a new one.
0075     objs(ii) = copy(objs(ii), nargout);
0076     if isnumeric(epochtime{1})
0077         objs(ii).utc_epoch_milli = epochtime{1};
0078     else
0079         error('### Epoch time must be numeric');
0080     end
0081 end
0082
0083 %%% Prepare output
0084 varargout{1} = objs;
0085
0086 end
0087
0088 %%%%%%%%%%%%%%
0089 %                               Local Functions                               %
0090 %%%%%%%%%%%%%%
0091
0092 %%%%%%%%%%%%%%
0093 %
0094 % FUNCTION:      getInfo

```

```
0095 %
0096 % DESCRIPTION: Get Info Object
0097 %
0098 % HISTORY:      11-07-07 M Hewitson
0099 %              Creation.
0100 %
0101 %%%%%%%%%%%
0102 %
0103 function ii = getInfo(varargin)
0104     if nargin == 1 && strcmpi(varargin{1}, 'None')
0105         sets = {};
0106         pl   = [];
0107     else
0108         sets = {'Default'};
0109         pl   = getDefaultPlist;
0110     end
0111     % Build info object
0112     ii = minfo(mfilename, 'time', '', utils.const.categories.internal, '$Id:
setEpochtime.m,v 1.5 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0113 end
0114 %%%%%%%%%%%
0115 %
0116 %
0117 % FUNCTION:      getDefaultPlist
0118 %
0119 % DESCRIPTION: Get Default Plist
0120 %
0121 % HISTORY:      11-07-07 M Hewitson
0122 %              Creation.
0123 %
0124 %%%%%%%%%%%
0125 %
0126 function plo = getDefaultPlist()
0127     plo = plist('utc_epoch_milli', '');
0128 end
0129
```

setTime_str

PURPOSE ^

SETTIME_STR Set the property 'time_str'.

SYNOPSIS ^

function varargout = setTime_str(varargin)

DESCRIPTION ^

```
SETTIME_STR Set the property 'time_str'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'time_str'.

CALL:      obj = obj.setTime_str('new time_str');
           obj = obj.setTime_str(plist('time_str', 'new time_str'));
           obj = setTime_str(obj, 'new time_str');

INPUTS:    obj - can be a vector, matrix, list, or a mix of them.
           pl  - to set the time_str with a plist specify only one plist with
                  only one key-word 'time_str'.

M-FILE INFO: Get information about this methods by calling
              >> time.getInfo('setTime_str')

              Get information about a specified set-plist by calling:
              >> time.getInfo('setTime_str', 'set')

VERSION:    $Id: setTime_str.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function tt = compute_epoch_t\(tt, time_str\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETTIME_STR Set the property 'time_str'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'time_str'.
0005 %
0006 % CALL:      obj = obj.setTime_str('new time_str');
```

```

0007 %           obj = obj.setTime_str(plist('time_str', 'new time_str'));
0008 %           obj = setTime_str(obj, 'new time_str');
0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %              pl - to set the time_str with a plist specify only one plist with
0012 %                   only one key-word 'time_str'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> time.getInfo('setTime_str')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> time.getInfo('setTime_str', 'set')
0019 %
0020 % VERSION:     $Id: setTime_str.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:      27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026 %
0027 function varargout = setTime_str(varargin)
0028 %
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034 %
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037 %
0038 %%% decide whether we modify the time-object, or create a new one.
0039 varargin{1} = copy(varargin{1}, nargout);
0040 %
0041 varargin{1} = compute_epoch_t(varargin{1}, varargin{2});
0042 varargout{1} = varargin{1};
0043 return
0044 end
0045 %
0046 %%% Normal call:
0047 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'time');
0048 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0049 %
0050 time_str = rest{1};
0051 if ~ischar(time_str)
0052     error('### Time string must be a string');
0053 end
0054 %
0055 if length(time_str) ~=1
0056 %
0057     if length(pls) == 1
0058         time_str = {};
0059         if nparams(pls) == 1 && isparam(pls, 'time_str')
0060 %
0061 %%% If the plist contains only one parameter with the key-word 'time_str'
0062 %%% then set 'time_str' to the value in the time object.
0063 time_str = find(pls, 'time_str');
0064 %
0065     else
0066         time_str = pls;
0067     end
0068 elseif length(pls) > 1
0069     error('### To set the ''time_str'' please specify only one plist')
0070 else
0071     error('### Please specify [only one] value.')
0072 end
0073 end
0074 %
0075 %%% Set the time_str
0076 for ii = 1:numel(objs)
0077 %%% decide whether we modify the time-object, or create a new one.
0078 objs(ii) = copy(objs(ii), nargout);
0079 objs(ii) = compute_epoch_t(objs(ii), time_str);
0080 end
0081 %
0082 %%% Prepare output
0083 varargout{1} = objs;
0084 %
0085 end
0086 %
0087 %%%%%%%%%%%%%%
0088 %                               Local Functions                               %
0089 %%%%%%%%%%%%%%
0090 %
0091 %%%%%%%%%%%%%%
0092 %

```

```
0093 % FUNCTION:      compute_epoch_t
0094 %
0095 % DESCRIPTION: Compute the epoch time from a time-string.
0096 %
0097 % HISTORY:        07-07-2008 M Diepholz
0098 %                  Creation.
0099 %
0100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0101
0102 function tt = compute_epoch_t(tt, time_str)
0103     [parse_java, parse_matlab] = time.parse_time_string(time_str);
0104
0105     tt.timeformat = parse_matlab;
0106
0107     t_format = java.text.SimpleDateFormat(parse_java);
0108     t_format.setTimeZone(tt.timezone);
0109     new_time = t_format.parse(time_str);
0110
0111     tt.utc_epoch_milli = new_time.getTime;
0112 end
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114 %
0115 % FUNCTION:      getInfo
0116 %
0117 % DESCRIPTION: Get Info Object
0118 %
0119 % HISTORY:        11-07-07 M Hewitson
0120 %                  Creation.
0121 %
0122 %
0123 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0124
0125 function ii = getInfo(varargin)
0126     if nargin == 1 && strcmpi(varargin{1}, 'None')
0127         sets = {};
0128         pl = [];
0129     else
0130         sets = {'Default'};
0131         pl = getDefaultPlist;
0132     end
0133     % Build info object
0134     ii = minfo(mfilename, 'time', '', utils.const.categories.internal, '$Id:
setTime_str.m,v 1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0135 end
0136 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0137 %
0138 % FUNCTION:      getDefaultPlist
0139 %
0140 % DESCRIPTION: Get Default Plist
0141 %
0142 % HISTORY:        11-07-07 M Hewitson
0143 %                  Creation.
0144 %
0145 %
0146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0147
0148 function plo = getDefaultPlist()
0149     plo = plist('time_str', '');
0150 end
0151
```

setTimeformat

PURPOSE ^

SETTIMEFORMAT Set the property 'timeformat'.

SYNOPSIS ^

function varargout = setTimeformat(varargin)

DESCRIPTION ^

```
SETTIMEFORMAT Set the property 'timeformat'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'timeformat'.

CALL:
    obj = obj.setTimeformat('new timeformat');
    obj = obj.setTimeformat(plist('timeformat', 'new timeformat'));
    obj = setTimeformat(obj, 'new timeformat');

INPUTS:
    obj - can be a vector, matrix, list, or a mix of them.
    pl  - to set the timeformat with a plist specify only one plist with
          only one key-word 'timeformat'.

M-FILE INFO: Get information about this methods by calling
    >> time.getInfo('setTimeformat')

            Get information about a specified set-plist by calling:
    >> time.getInfo('setTimeformat', 'set')

VERSION:
    $Id: setTimeformat.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function val_str = matlab_format2str\(val\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETTIMEFORMAT Set the property 'timeformat'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'timeformat'.
0005 %
0006 % CALL:
    obj = obj.setTimeformat('new timeformat');
```

```

0007 %           obj = obj.setTimeformat(plist('timeformat', 'new timeformat'));
0008 %           obj = setTimeformat(obj, 'new timeformat');
0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl - to set the timeformat with a plist specify only one plist with
0012 %                   only one key-word 'timeformat'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %               >> time.getInfo('setTimeformat')
0016 %
0017 %               Get information about a specified set-plist by calling:
0018 %               >> time.getInfo('setTimeformat', 'set')
0019 %
0020 % VERSION:     $Id: setTimeformat.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026 %
0027 function varargout = setTimeformat(varargin)
0028 %
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034 %
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037 %
0038 %%% decide whether we modify the time-object, or create a new one.
0039 varargin{1} = copy(varargin{1}, nargout);
0040 %
0041 if isnumeric(varargin{2})
0042     varargin{1}.timeformat = matlab_format2str(varargin{2});
0043 else
0044     varargin{1}.timeformat = varargin{2};
0045 end
0046 varargout{1} = varargin{1};
0047 return
0048 end
0049 %
0050 %%% Normal call:
0051 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'time');
0052 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0053 %
0054 timeformat = rest;
0055 %
0056 if length(timeformat) ~=1
0057 %
0058 if length(pls) == 1
0059     timeformat = {};
0060     if nparams(pls) == 1 && isparam(pls, 'timeformat')
0061 %
0062 %%% If the plist contains only one parameter with the key-word 'timeformat'
0063 %%% then set 'timeformat' to the value in the time object.
0064 timeformat{1} = find(pls, 'timeformat');
0065 %
0066 else
0067     timeformat{1} = pls;
0068 end
0069 elseif length(pls) > 1
0070     error('### To set the 'timeformat' please specify only one plist')
0071 else
0072     error('### Please specify [only one] value.')
0073 end
0074 end
0075 %
0076 %%% Set the timeformat
0077 for ii = 1:numel(objs)
0078 %%% decide whether we modify the time-object, or create a new one.
0079     objs(ii) = copy(objs(ii), nargout);
0080 %
0081 if isnumeric(timeformat{1})
0082     objs(ii).timeformat = matlab_format2str(timeformat{1});
0083 elseif ischar(timeformat{1})
0084     objs(ii).timeformat = timeformat{1};
0085 else
0086     error('### Timeformat must be a string');
0087 end
0088 end
0089 %
0090 %%% Prepare output
0091 varargout{1} = objs;
0092 %

```



```
0093 end
0094
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096 %                               Local Functions                               %
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100 %
0101 % FUNCTION:      matlab_format2str
0102 %
0103 % DESCRIPTION: Convert the standard MATLAB Date format definitions into a string
0104 %
0105 % HISTORY:      11-07-07 M Diepholz
0106 %               Creation.
0107 %
0108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109
0110 function val_str = matlab_format2str(val)
0111
0112     switch val
0113     case 0
0114         val_str = 'dd-mmm-yyyy HH:MM:SS';
0115     case 1
0116         val_str = 'dd-mmm-yyyy';
0117     case 2
0118         val_str = 'mm/dd/yy';
0119     case 3
0120         val_str = 'mmm';
0121     case 4
0122         val_str = 'm';
0123     case 5
0124         val_str = 'mm';
0125     case 6
0126         val_str = 'mm/dd';
0127     case 7
0128         val_str = 'dd';
0129     case 8
0130         val_str = 'ddd';
0131     case 9
0132         val_str = 'd';
0133     case 10
0134         val_str = 'yyyy';
0135     case 11
0136         val_str = 'yy';
0137     case 12
0138         val_str = 'mmmyy';
0139     case 13
0140         val_str = 'HH:MM:SS';
0141     case 14
0142         val_str = 'HH:MM:SS PM';
0143     case 15
0144         val_str = 'HH:MM';
0145     case 16
0146         val_str = 'HH:MM PM';
0147     case 19
0148         val_str = 'dd/mm';
0149     case 20
0150         val_str = 'dd/mm/yy';
0151     case 21
0152         val_str = 'mmm.dd,yyyy HH:MM:SS';
0153     case 22
0154         val_str = 'mmm.dd,yyyy';
0155     case 23
0156         val_str = 'mm/dd/yyyy';
0157     case 24
0158         val_str = 'dd/mm/yyyy';
0159     case 25
0160         val_str = 'yy/mm/dd';
0161     case 26
0162         val_str = 'yyyy/mm/dd';
0163     case 28
0164         val_str = 'mmmyyyy';
0165     case 29
0166         val_str = 'yyyy-mm-dd';
0167     case 31
0168         val_str = 'yyyy-mm-dd HH:MM:SS';
0169     otherwise
0170         error('### The format number [%d] is not supported', val);
0171     end
0172 end
0173
0174 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0175 %
0176 % FUNCTION:      getInfo
0177 %
0178 % DESCRIPTION: Get Info Object
```



```
0179 %
0180 % HISTORY:      11-07-07 M Hewitson
0181 %              Creation.
0182 %
0183 %%%%%%%%%%%%%%
0184
0185 function ii = getInfo(varargin)
0186     if nargin == 1 && strcmpi(varargin{1}, 'None')
0187         sets = {};
0188         pl   = [];
0189     else
0190         sets = {'Default'};
0191         pl   = getDefaultPlist;
0192     end
0193     % Build info object
0194     ii = minfo(mfilename, 'time', '', utils.const.categories.internal, '$Id:
setTimeformat.m,v 1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0195 end
0196
0197 %%%%%%%%%%%%%%
0198 %
0199 % FUNCTION:      getDefaultPlist
0200 %
0201 % DESCRIPTION:   Get Default Plist
0202 %
0203 % HISTORY:      11-07-07 M Hewitson
0204 %              Creation.
0205 %
0206 %%%%%%%%%%%%%%
0207
0208 function plo = getDefaultPlist()
0209     plo = plist('timeformat', '');
0210 end
0211
```

setTimezone

PURPOSE ^

SETTIMEZONE Set the property 'timezone'.

SYNOPSIS ^

function varargout = setTimezone(varargin)

DESCRIPTION ^

SETTIMEZONE Set the property 'timezone'.
%%%

DESCRIPTION: Set the property 'timezone'.

CALL:
obj = obj.setTimezone('new timezone');
obj = obj.setTimezone(plist('timezone', 'new timezone'));
obj = setTimezone(obj, 'new timezone');

INPUTS:
obj - can be a vector, matrix, list, or a mix of them.
pl - to set the timezone with a plist specify only one plist with
only one key-word 'timezone'.

M-FILE INFO: Get information about this methods by calling
>> time.getInfo('setTimezone')

Get information about a specified set-plist by calling:
>> time.getInfo('setTimezone', 'set')

VERSION: \$Id: setTimezone.m,v 1.3 2008/09/04 15:29:31 ingo Exp \$

HISTORY: 27-05-2008 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETTIMEZONE Set the property 'timezone'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'timezone'.
0005 %
0006 % CALL:      obj = obj.setTimezone('new timezone');
0007 %           obj = obj.setTimezone(plist('timezone', 'new timezone'));
0008 %           obj = setTimezone(obj, 'new timezone');
```

```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl  - to set the timezone with a plist specify only one plist with
0012 %                   only one key-word 'timezone'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> time.getInfo('setTimezone')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> time.getInfo('setTimezone', 'set')
0019 %
0020 % VERSION:     $Id: setTimezone.m,v 1.3 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %              Creation
0024 %
0025 %%%%%%%%%%%%%%%
0026
0027 function varargout = setTimezone(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037
0038     %%% decide whether we modify the time-object, or create a new one.
0039     varargin{1} = copy(varargin{1}, nargout);
0040
0041     available_IDs = cell(java.util.TimeZone.getAvailableIDs);
0042     if ischar(varargin{2}) && ismember(varargin{2}, available_IDs)
0043         varargin{1}.timezone = java.util.TimeZone.getTimeZone(varargin{2});
0044     elseif ischar(varargin{2})
0045         error('### Unknown timezone [%s]', varargin{2});
0046     else
0047         varargin{1}.timezone = varargin{2};
0048     end
0049     varargout{1} = varargin{1};
0050     return
0051 end
0052
0053 %%% Normal call:
0054 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'time');
0055 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0056
0057 timezone = rest;
0058
0059 if length(timezone) ~= 1
0060
0061     if length(pls) == 1
0062         timezone = {};
0063         if nparams(pls) == 1 && isparam(pls, 'timezone')
0064
0065             %%% If the plist contains only one parameter with the key-word 'timezone'
0066             %%% then set 'timezone' to the value in the time object.
0067             timezone{1} = find(pls, 'timezone');
0068
0069         else
0070             timezone{1} = pls;
0071         end
0072     elseif length(pls) > 1
0073         error('### To set the 'timezone' please specify only one plist')
0074     else
0075         error('### Please specify [only one] value.')
0076     end
0077 end
0078
0079 %%% Set the timezone
0080 available_IDs = cell(java.util.TimeZone.getAvailableIDs);
0081 if ischar(timezone{1}) && ismember(timezone{1}, available_IDs)
0082     timezone = java.util.TimeZone.getTimeZone(timezone{1});
0083 elseif ischar(timezone{1})
0084     error('### Unknown timezone [%s]', timezone{1});
0085 else
0086     timezone = timezone{1};
0087 end
0088
0089 for ii = 1:numel(objs)
0090     %%% decide whether we modify the time-object, or create a new one.
0091     objs(ii) = copy(objs(ii), nargout);
0092     objs(ii).timezone = timezone;
0093 end
0094

```

```
0095    %% Prepare output
0096    varargin{1} = objs;
0097
0098 end
0099
0100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0101 %                                     Local Functions                                     %
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %
0106 % FUNCTION:      getInfo
0107 %
0108 % DESCRIPTION:  Get Info Object
0109 %
0110 % HISTORY:      11-07-07 M Hewitson
0111 %               Creation.
0112 %
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114
0115 function ii = getInfo(varargin)
0116     if nargin == 1 && strcmpi(varargin{1}, 'None')
0117         sets = {};
0118         pl   = [];
0119     else
0120         sets = {'Default'};
0121         pl   = getDefaultPlist;
0122     end
0123     % Build info object
0124     ii = minfo(mfilename, 'time', '', utils.const.categories.internal, '$Id:
setTimezone.m,v 1.3 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0125 end
0126
0127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0128 %
0129 % FUNCTION:      getDefaultPlist
0130 %
0131 % DESCRIPTION:  Get Default Plist
0132 %
0133 % HISTORY:      11-07-07 M Hewitson
0134 %               Creation.
0135 %
0136 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0137
0138 function plo = getDefaultPlist()
0139     plo = plist('timezone', '');
0140 end
0141
```

char

PURPOSE ^

CHAR convert a time object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a time object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a time object into a string.

CALL:      string = char(time)

M-FILE INFO: Get information about this methods by calling
              >> time.getInfo('char')

              Get information about a specified set-plist by calling:
              >> time.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.12 2008/09/04 15:29:31 ingo Exp $

HISTORY:    23-07-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [display](#) DISPLAY overloads display functionality for time objects.
-  [epoch2str](#) EPOCH2STR Returns computes the time sting of the epoch time stored in 'utc_epoch_milli' and 'created'.
-  [format](#) FORMAT Returns the time in specified format.
-  [string](#) STRING writes a command string that can be used to recreate the input time object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a time object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a time object into a string.
0005 %
0006 % CALL:      string = char(time)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> time.getInfo('char')
0010 %
```

```
0011 %           Get information about a specified set-plist by calling:
0012 %           >> time.getInfo('char', 'set')
0013 %
0014 % VERSION:      $Id: char.m,v 1.12 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:       23-07-2007 Diepholz
0017 %                 Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 % Collect all time-objects
0030 objs = utils.helper.collect_objects(varargin(:), 'time');
0031
0032 % go through time-object
0033 pstr = '';
0034 for jj=1:numel(objs)
0035     pstr = [pstr time.epoch2str(objs(jj)) ' '];
0036 end
0037
0038 varargout{1} = pstr(1:end-2);
0039 end
0040
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %                               Local Functions
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:       getInfo
0048 %
0049 % DESCRIPTION:    Get Info Object
0050 %
0051 % HISTORY:        11-07-07 M Hewitson
0052 %                 Creation.
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl    = [];
0060     else
0061         sets = {'Default'};
0062         pl    = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'time', '', utils.const.categories.output, '$Id: char.m,v 1.12
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:       getDefaultPlist
0071 %
0072 % DESCRIPTION:    Get Default Plist
0073 %
0074 % HISTORY:        11-07-07 M Hewitson
0075 %                 Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082
```

display

PURPOSE ^

DISPLAY overloads display functionality for time objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for time objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for time objects.

CALL:      txt = display(time)

INPUT:     time - time object

OUTPUT:    txt - cell array with strings to display the time object

M-FILE INFO: Get information about this methods by calling
              >> time.getInfo('display')

              Get information about a specified set-plist by calling:
              >> time.getInfo('display', 'set')

VERSION:   $Id: display.m,v 1.19 2008/09/04 15:29:31 ingo Exp $

HISTORY:   23-07-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a time object into a string.
-  [epoch2str](#) EPOCH2STR Returns computes the time sting of the epoch time stored in 'utc_epoch_milli' and 'created'.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for time objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for time objects.
0005 %
0006 % CALL:      txt = display(time)
0007 %
0008 % INPUT:     time - time object
0009 %
```

```

0010 % OUTPUT:      txt - cell array with strings to display the time object
0011 %
0012 % M-FILE INFO:  Get information about this methods by calling
0013 %               >> time.getInfo('display')
0014 %
0015 %               Get information about a specified set-plist by calling:
0016 %               >> time.getInfo('display', 'set')
0017 %
0018 % VERSION:      $Id: display.m,v 1.19 2008/09/04 15:29:31 ingo Exp $
0019 %
0020 % HISTORY:      23-07-2007 Diepholz
0021 %               Creation
0022 %
0023 %%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027 %%% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 % Collect all time-objects
0034 tt = utils.helper.collect_objects(varargin(:), 'time');
0035
0036 txt = {};
0037
0038 for ii = 1:numel(tt)
0039     t = tt(ii);
0040
0041     b_banner = sprintf('----- time %02d -----', ii);
0042     txt{end+1} = b_banner;
0043     txt{end+1} = ' ';
0044
0045     fields = fieldnames(t);
0046     max_length = length(fields{1});
0047     for jj = 2:length(fields)
0048         if length(fields{jj}) > max_length
0049             max_length = length(fields{jj});
0050         end
0051     end
0052
0053     for jj = 1:length(fields)
0054         field = fields{jj};
0055
0056         str_field = [];
0057         str_field(1:max_length-length(field)) = ' ';
0058         str_field = [field str_field];
0059
0060         %%% Special case: ----- time_str -----
0061         %%% Compute the property time_str
0062         if strcmp(field, 'time_str')
0063             time_str = time.epoch2str(t);
0064             txt{end+1} = sprintf('%s: %s', str_field, time_str);
0065             continue
0066         end
0067
0068         % Add before the fields 'created' and 'version' a blank line
0069         if strcmp(field, 'created')
0070             txt{end+1} = ' ';
0071             end
0072
0073         %%% Special case: ----- created -----
0074         %%% Compute the property created
0075         if strcmp(field, 'created')
0076             [time_str, created_str] = epoch2str(t);
0077             txt{end+1} = sprintf('%s: %s', str_field, created_str);
0078             continue
0079         end
0080
0081         % Display: Number
0082         if isnumeric(t.(field))
0083             txt{end+1} = sprintf('%s: %s', str_field, num2str(t.(field)));
0084
0085         % Display: Strings
0086         elseif ischar(t.(field))
0087             txt{end+1} = sprintf('%s: %s', str_field, t.(field));
0088
0089         % Display: Logicals
0090         elseif islogical(t.(field))
0091             if t.(field) == true
0092                 txt{end+1} = sprintf('%s: true', str_field);
0093             else
0094                 txt{end+1} = sprintf('%s: false', str_field);
0095             end
0096         end
0097     end
0098 end

```



```

0096
0097 % Display: Objects
0098 elseif isobject(t.(field))
0099     if isa(t.(field), 'timeformat') || isa(t.(field), 'plist')
0100         txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field)));
0101     else
0102         txt{end+1} = sprintf ('%s: %s class',str_field, class(t.(field)));
0103     end
0104
0105 % Display: Java Objects
0106 elseif isjava(t.(field))
0107     if strcmp(class(t.(field)), 'sun.util.calendar.ZoneInfo')
0108         EE = java.text.SimpleDateFormat('Z');
0109         EE.setTimeZone(t.(field))
0110         zone_id = t.(field).getID;
0111         if strcmp(zone_id, 'UTC')
0112             txt{end+1} = sprintf ('%s: %s',str_field, char(t.(field).getID));
0113         else
0114             txt{end+1} = sprintf ('%s: %s (UTC%s)',str_field, char(t.(field).getID),
char(EE.format(t.utc_epoch_milli)));
0115         end
0116     else
0117         error ('### Unknown java class %s', class(t.(field)))
0118     end
0119
0120 else
0121     error ('### Please define the output for this property %s', field)
0122 end
0123
0124 end
0125
0126 e_banner(1:length(b_banner)) = '-';
0127 txt{end+1} = e_banner;
0128
0129 txt{end+1} = ' ';
0130 txt{end+1} = ' ';
0131 end
0132
0133 if nargout == 0
0134     for ii=1:length(txt)
0135         disp(txt{ii});
0136     end
0137 end
0138
0139 varargout{1} = txt;
0140 end
0141
0142 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0143 % Local Functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0144 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0145 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0147 %
0148 % FUNCTION:      getInfo
0149 %
0150 % DESCRIPTION:   Get Info Object
0151 %
0152 % HISTORY:       11-07-07 M Hewitson
0153 %                 Creation.
0154 %
0155 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0156
0157 function ii = getInfo(varargin)
0158     if nargin == 1 && strcmpi(varargin{1}, 'None')
0159         sets = {};
0160         pl = [];
0161     else
0162         sets = {'Default'};
0163         pl = getDefaultPlist;
0164     end
0165     % Build info object
0166     ii = minfo(mfilename, 'time', '', utils.const.categories.output, '$Id: display.m,v
1.19 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0167 end
0168
0169 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0170 %
0171 % FUNCTION:      getDefaultPlist
0172 %
0173 % DESCRIPTION:   Get Default Plist
0174 %
0175 % HISTORY:       11-07-07 M Hewitson
0176 %                 Creation.
0177 %
0178 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0179

```

```
0180 function plo = getDefaultPlist()  
0181     plo = plist();  
0182 end  
0183
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

format

PURPOSE ^

FORMAT Returns the time in specified format.

SYNOPSIS ^

function varargout = format(varargin)

DESCRIPTION ^

```
FORMAT Returns the time in specified format.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FORMAT Returns the time in specified format.

CALL:      string = format(tt, format_str);
           string = format(tt, 'HH:MM:SS');

M-FILE INFO: Get information about this methods by calling
             >> time.getInfo('format')

             Get information about a specified set-plist by calling:
             >> time.getInfo('format', 'set')

VERSION:    $Id: format.m,v 1.11 2008/09/04 15:29:31 ingo Exp $

HISTORY:    23-07-2007 Diepholz
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a time object into a string.

This function is called by:

 [time](#) TIME time object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % FORMAT Returns the time in specified format.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FORMAT Returns the time in specified format.
0005 %
0006 % CALL:      string = format(tt, format_str);
0007 %           string = format(tt, 'HH:MM:SS');
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %             >> time.getInfo('format')
0011 %
0012 %             Get information about a specified set-plist by calling:
0013 %             >> time.getInfo('format', 'set')
```

```
0014 %
0015 % VERSION:      $Id: format.m,v 1.11 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      23-07-2007 Diepholz
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021 %
0022 function varargout = format(varargin)
0023 %
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029 %
0030 % Collect all time-objects
0031 [tt, invars, rest] = utils.helper.collect_objects(varargin(:), 'time');
0032 %
0033 if isempty(rest)
0034     format_str = tt(1).timeformat;
0035 else
0036     format_str = rest{1};
0037 end
0038 %
0039 tt = tt.setTimeformat(format_str);
0040 %
0041 varargout{1} = char(tt);
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                                Local Functions                                %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060 if nargin == 1 && strcmpi(varargin{1}, 'None')
0061     sets = {};
0062     pl    = [];
0063 else
0064     sets = {'Default'};
0065     pl    = getDefaultPlist;
0066 end
0067 % Build info object
0068 ii = minfo(mfilename, 'time', '', utils.const.categories.output, '$Id: format.m,v 1.11
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist();
0084 end
0085 %
```

pz

PURPOSE ^

PZ is the ltpda class that provides a common definition of poles and zeros.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

PZ is the ltpda class that provides a common definition of poles and zeros.
%%%

DESCRIPTION: PZ is the ltpda class that provides a common definition of poles and zeros.

CONSTRUCTORS:

```
p = pz(f)      % specify frequency
p = pz(f,q)    % specify frequency and Q
p = pz(c)      % create with complex representation
```

SUPER CLASSES: ltpda_nuo < ltpda_obj

PROPERTIES:

```
Inherit Properties (read only)
  version - cvs-version string.

Protected Properties (read only)
  f        - frequency of pole/zero
  q        - quality factor of pole/zero
  ri       - complex representation of pole/zero
```

PZ METHODS:

```
Defined Abstract methods:
char      - returns one character string which represents the object
copy      - copies an object
display   - displays an object
update_struct - updates a object structure to the current tbx-version

Public methods:
cz2iir - return a,b IIR filter coefficients for a complex zero designed
        using the bilinear transform.
cp2iir - return a,b IIR filter coefficients for a complex pole designed
        using the bilinear transform.
rp2iir - return a,b coefficients for a real pole designed using the
        bilinear transform.
rz2iir - return a,b IIR filter coefficients for a real zero designed using
        the bilinear transform.

Protected methods:
setF      - set the frequency property
setQ      - set the quality factor
setRI     - set the complex form of the pole/zero
ri2fq     - convert complex pole/zero into frequency/Q pole/zero representation.
fq2ri     - convert frequency/Q pole/zero representation
```

M-FILE INFO: The following call returns an minfo object that contains information about the pz constructor:
 >> info = pz.getInfo
or >> info = pz.getInfo('pz')

You can get information about class methods by calling:
 >> info = pz.getInfo(method)
e.g. >> info = pz.getInfo('eq')

You can also restrict the sets of parameters contained in the minfo object by calling:
 >> info = pz.getInfo(method, set)

```
e.g. >> info = pz.getInfo('param', 'Default')
VERSION:      $Id: pz.m,v 1.23 2008/09/03 16:36:02 hewitson Exp $
HISTORY:      19-05-2008 Diepholz
               Creation.
SEE ALSO:     ltpda_obj, ltpda_nuo, pzmodel
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of pz objects depending of the second input
-  [cp2iir](#) CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.
-  [cz2iir](#) CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
-  [fq2ri](#) FQ2RI Convert frequency/Q pole/zero representation
-  [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.
-  [resp](#) RESP returns the complex response of the pz object.
-  [ri2fq](#) RI2FQ Convert complex pole/zero into frequency/Q pole/zero representation.
-  [rp2iir](#) RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
-  [rz2iir](#) RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.
-  [setF](#) SETF Set the property 'f'
-  [setQ](#) SETQ Set the property 'q'
-  [setRI](#) SETRI Set the property 'ri' and computes 'f' and 'q'
-  [string](#) STRING writes a command string that can be used to recreate the input pz object.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [copy](#) COPY Make copy of pz objects depending of the second input
-  [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

SUBFUNCTIONS ^

-  [function obj = set.f\(obj, val\)](#)
-  [function obj = set.q\(obj, val\)](#)
-  [function obj = set.ri\(obj, val\)](#)
-  [function obj = pz\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 % PZ is the ltpda class that provides a common definition of poles and zeros.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PZ is the ltpda class that provides a common definition of
0005 %               poles and zeros.
0006 %
0007 % CONSTRUCTORS:
0008 %
0009 %           p = pz(f)    % specify frequency
0010 %           p = pz(f,q) % specify frequency and Q
```

```

0011 %           p = pz(c)    % create with complex representation
0012 %
0013 %
0014 % SUPER CLASSES: ltpda_nuo < ltpda_obj
0015 %
0016 % PROPERTIES:
0017 %
0018 %     Inherit Properties (read only)
0019 %         version - cvs-version string.
0020 %
0021 %     Protected Properties (read only)
0022 %         f         - frequency of pole/zero
0023 %         q         - quality factor of pole/zero
0024 %         ri        - complex representation of pole/zero
0025 %
0026 % PZ METHODS:
0027 %
0028 %     Defined Abstract methods:
0029 %         char       - returns one character string which represents the object
0030 %         copy       - copies an object
0031 %         display    - displays an object
0032 %         update_struct - updates a object structure to the current tbx-version
0033 %
0034 %     Public methods:
0035 %         cz2iir - return a,b IIR filter coefficients for a complex zero designed
0036 %                 using the bilinear transform.
0037 %         cp2iir - return a,b IIR filter coefficients for a complex pole designed
0038 %                 using the bilinear transform.
0039 %         rp2iir - return a,b coefficients for a real pole designed using the
0040 %                 bilinear transform.
0041 %         rz2iir - return a,b IIR filter coefficients for a real zero designed using
0042 %                 the bilinear transform.
0043 %
0044 %     Protected methods:
0045 %         setF       - set the frequency property
0046 %         setQ       - set the quality factor
0047 %         setRI      - set the complex form of the pole/zero
0048 %         ri2fq      - convert complex pole/zero into frequency/Q pole/zero representation.
0049 %         fq2ri      - convert frequency/Q pole/zero representation
0050 %
0051 % M-FILE INFO: The following call returns an minfo object that contains
0052 %               information about the pz constructor:
0053 %               >> info = pz.getInfo
0054 %               or   >> info = pz.getInfo('pz')
0055 %
0056 %               You can get information about class methods by calling:
0057 %               >> info = pz.getInfo(method)
0058 %               e.g. >> info = pz.getInfo('eq')
0059 %
0060 %               You can also restrict the sets of parameters contained in
0061 %               the minfo object by calling:
0062 %               >> info = pz.getInfo(method, set)
0063 %               e.g. >> info = pz.getInfo('param', 'Default')
0064 %
0065 % VERSION:      $Id: pz.m,v 1.23 2008/09/03 16:36:02 hewitson Exp $
0066 %
0067 % HISTORY:      19-05-2008 Diepholz
0068 %                Creation.
0069 %
0070 % SEE ALSO: ltpda_obj, ltpda_nuo, pzmodel
0071 %
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073
0074 classdef pz < ltpda_nuo
0075
0076 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077 %                                Property definition                                %
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079
0080 %----- Public (read/write) Properties -----
0081 properties
0082 end
0083
0084 %----- Protected read-only Properties -----
0085 properties (SetAccess = protected)
0086     f = NaN;
0087     q = NaN;
0088     ri = NaN;
0089     version = '$Id: pz.m,v 1.23 2008/09/03 16:36:02 hewitson Exp $';
0090 end
0091
0092 %----- Private Properties -----
0093 properties (GetAccess = protected, SetAccess = protected)
0094 end
0095
0096 %----- Abstract Properties -----

```

```

0097 properties (Abstract = true, SetAccess = protected)
0098 end
0099
0100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0101 %                                Check property setting                                %
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103
0104 methods
0105     function obj = set.f(obj, val)
0106         if ~isnumeric(val) || isempty(val) || ~isreal(val)
0107             error('### The value for the property ''f'' must be a real positive number');
0108         end
0109         obj.f = val;
0110     end
0111     function obj = set.q(obj, val)
0112         if ~isnumeric(val) || isempty(val) || ~isreal(val)
0113             error('### The value for the property ''q'' must be a real positive number');
0114         end
0115         obj.q = val;
0116     end
0117     function obj = set.ri(obj, val)
0118         if ~isnumeric(val) || isempty(val)
0119             error('### The value for the property ''ri'' must be a number');
0120         end
0121         obj.ri = val;
0122     end
0123 end
0124
0125 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0126 %                                Constructor                                %
0127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0128
0129 methods
0130     function obj = pz(varargin)
0131
0132         %% Call superclass
0133         obj = obj@ltpda_nuo(varargin{:});
0134
0135         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0136         Set default values %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0137
0138         if nargin == 0
0139             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0140             no inputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141
0142         elseif nargin == 1
0143             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0144             one input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0145
0146             if isa(varargin{1}, 'pz')
0147                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0148                 Copy ltpda_obj Object %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0149                 p1 = pz(pz-object) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0150                 obj = copy(varargin{1}, 1);
0151
0152             elseif isnumeric(varargin{1})
0153                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0154                 real or complex pole/zero %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0155                 if length(varargin{1}) == 1
0156                     if isreal(varargin{1})
0157                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0158                         p1 = pz(3) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0159                         obj.setF(varargin{1});
0160                         obj.setQ(NaN);
0161                     else
0162                         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0163                         p1 = pz(1+3i) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0164                         obj.setRI(varargin{1});
0165                     end
0166                 elseif length(varargin{1}) == 2
0167                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0168                     p1 = pz([1 2]) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0169                     obj.setF(varargin{1}(1));
0170                     obj.setQ(varargin{1}(2));
0171                 else
0172                     error('### Unknown constructor for the input %s', mat2str(varargin{1}));
0173                 end
0174
0175             elseif isa(varargin{1}, 'plist')
0176                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0177                 p1 = pz(plist-object) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0178                 pl = varargin{1};
0179                 f = find(pl, 'f');
0180                 q = find(pl, 'q');
0181                 ri = find(pl, 'ri');
0182                 if ~isempty(f)
0183                     obj.setF(f);
0184                 if ~isempty(q)
0185                     obj.setQ(q);
0186                 end
0187                 elseif ~isempty(ri)

```



```

0183         obj.setRI(ri);
0184     else
0185         error('### Unknown plist constructor');
0186     end
0187
0188     elseif isstruct(varargin{1})
0189         %%%%%%%%%%% p1 = pz(structure) %%%%%%%%%%%
0190         %%% Set properties which are declared in this class
0191         obj.f = varargin{1}.f;
0192         obj.q = varargin{1}.q;
0193         obj.ri = varargin{1}.ri;
0194         obj.version = varargin{1}.version;
0195
0196     elseif iscell(varargin{1})
0197         %%%%%%%%%%% p1 = pz([1 2], 1+2i, 3) %%%%%%%%%%%
0198         obj(1:numel(varargin{1})) = pz();
0199         for kk = 1:numel(varargin{1})
0200             obj(kk) = pz(varargin{1}{kk});
0201         end
0202
0203     else
0204         error('### Unknown call of the pz constructor. ');
0205     end
0206
0207     elseif nargin == 2
0208         %%%%%%%%%%% one input %%%%%%%%%%%
0209         %%%%%%%%%%% one input %%%%%%%%%%%
0210         %%%%%%%%%%% one input %%%%%%%%%%%
0211
0212         %%%%%%%%%%% obj = pz(f, q) %%%%%%%%%%%
0213         % f,q constructor
0214         obj.setF(varargin{1});
0215         obj.setQ(varargin{2});
0216     else
0217         error('### Unknown number of arguments. ');
0218     end
0219 end
0220
0221 end
0222
0223 %%%%%%%%%%%
0224 %
0225 % Methods (public) %
0226 %%%%%%%%%%%
0227
0228 methods
0229     varargout = cp2iir(varargin)
0230     varargout = cz2iir(varargin)
0231     varargout = rz2iir(varargin)
0232     varargout = rp2iir(varargin)
0233 end
0234
0235 %%%%%%%%%%%
0236 %
0237 % Methods (protected) %
0238 %%%%%%%%%%%
0239 methods (Access = protected)
0240     ii = setF(ii, val)
0241     ii = setQ(ii, val)
0242     ii = setRI(ii, val)
0243 end
0244
0245 %%%%%%%%%%%
0246 %
0247 % Methods (public) %
0248 %%%%%%%%%%%
0249 methods
0250     varargout = copy(varargin)
0251     [f,r] = resp(varargin)
0252     varargout = string(varargin)
0253 end
0254
0255 %%%%%%%%%%%
0256 %
0257 % Methods (protected, static) %
0258 %%%%%%%%%%%
0259 methods (Access = protected, Static)
0260     [f0, q] = ri2fq(c);
0261     ri = fq2ri(f0, Q);
0262 end
0263
0264 %%%%%%%%%%%
0265 %
0266 % Methods (static) %
0267 %%%%%%%%%%%
0268

```

```
0269 methods (Static)
0270
0271     varargout = update_struct(varargin);
0272
0273     function out = VEROUT()
0274         out = '$Id: pz.m,v 1.23 2008/09/03 16:36:02 hewitson Exp $';
0275     end
0276
0277     function ii = getInfo(varargin)
0278         ii = utils.helper.generic_getInfo(varargin{:}, 'pz');
0279     end
0280
0281     function out = SETS()
0282         out = {'Default'};
0283     end
0284
0285     function out = getDefaultPlist(set)
0286         switch set
0287             case 'Default'
0288                 out = plist();
0289             otherwise
0290                 out = plist();
0291         end
0292     end
0293
0294 end
0295
0296 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0297 %                                Methods (Static, Private)                                %
0298 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0299
0300 methods (Static = true, Access = private)
0301 end
0302
0303 end % End classdef
```

copy

PURPOSE ^

COPY Make copy of pz objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of pz objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of pz objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> pz.getInfo('copy')

VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

This function is called by:

-  [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.
-  [resp](#) RESP returns the complex response of the pz object.
-  [setF](#) SETF Set the property 'f'
-  [setQ](#) SETQ Set the property 'q'
-  [setRi](#) SETRI Set the property 'ri' and computes 'f' and 'q'

SOURCE CODE ^

```
0001 % COPY Make copy of pz objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of pz objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> pz.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function varargout = copy(old, deepcopy)
```

```
0019
0020
0021 if deepcopy
0022     ps = properties('pz');
0023     s = size(old);
0024     obj(s(1),s(2)) = pz;
0025     for kk=1:numel(old)
0026         utils.helper.msg(utils.const.msg.OPROC2, 'copying pz');
0027         for jj=1:length(ps)
0028             obj(kk).(ps{jj}) = old(kk).(ps{jj});
0029         end
0030         % reset on copy
0031         obj(kk).version = pz.VEROUT;
0032     end
0033 else
0034     obj = old;
0035 end
0036 varargout{1} = obj;
0037 end
0038
0039
```

cp2iir

PURPOSE ^

CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.

SYNOPSIS ^

function varargout = cp2iir(varargin)

DESCRIPTION ^

```
CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear
transform.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CP2IIR Return a,b IIR filter coefficients for a complex pole
designed using the bilinear transform.

CALL:      [a,b] = cp2iir(p, fs)

REMARK:    This is just a helper function. This function should only be
called from class functions.

INPUT:     p - pole object
           fs - the sample rate for the filter

VERSION:   $Id: cp2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:   03-04-2007 M Hewitson
           Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the
0002 % bilinear transform.
0003 %
0004 % DESCRIPTION: CP2IIR Return a,b IIR filter coefficients for a complex pole
0005 % designed using the bilinear transform.
0006 %
0007 % CALL:      [a,b] = cp2iir(p, fs)
0008 %
0009 % REMARK:    This is just a helper function. This function should only be
0010 % called from class functions.
0011 %
0012 % INPUT:     p - pole object
0013 %           fs - the sample rate for the filter
0014 %
```

```
0015 % VERSION:      $Id: cp2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:       03-04-2007 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = cp2iir(varargin)
0023
0024     %% Check if this is a call for parameters
0025     if utils.helper.isinfocall(varargin{:})
0026         varargout{1} = getInfo(varargin{3});
0027         return
0028     end
0029
0030     p = varargin{1};
0031     fs = varargin{2};
0032
0033     f0 = p.f;
0034     q = p.q;
0035
0036     w0 = f0*2*pi;
0037     w02 = w0^2;
0038
0039     k = (q*w02 + 4*q*fs*fs + 2*w0*fs) / (q*w02);
0040     b(1) = 1;
0041     b(2) = (2*w02-8*fs*fs) / (k*w02);
0042     b(3) = (q*w02 + 4*q*fs*fs - 2*w0*fs) / (k*q*w02);
0043
0044     a(1) = 1/k;
0045     a(2) = -2/k;
0046     a(3) = -1/k;
0047     a = a*-2;
0048
0049     varargout{1} = a;
0050     varargout{2} = b;
0051 end
0052
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054 %                               Local Functions                               %
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 % FUNCTION:      getInfo
0060 %
0061 % DESCRIPTION:   Get Info Object
0062 %
0063 % HISTORY:       11-07-07 M Hewitson
0064 %               Creation.
0065 %
0066 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0067
0068 function ii = getInfo(varargin)
0069     if nargin == 1 && strcmpi(varargin{1}, 'None')
0070         sets = {};
0071         pl = [];
0072     else
0073         sets = {'Default'};
0074         pl = getDefaultPlist;
0075     end
0076     % Build info object
0077     ii = minfo(mfilename, 'pz', '', utils.const.categories.internal, '$Id: cp2iir.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0078 end
0079
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 % FUNCTION:      getDefaultPlist
0083 %
0084 % DESCRIPTION:   Get Default Plist
0085 %
0086 % HISTORY:       11-07-07 M Hewitson
0087 %               Creation.
0088 %
0089 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090
0091 function plo = getDefaultPlist()
0092     plo = plist();
0093 end
0094
```

cz2iir

PURPOSE ^

CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.

SYNOPSIS ^

function varargout = cz2iir(varargin)

DESCRIPTION ^

```
CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CZ2IIR return a,b IIR filter coefficients for a complex zero
              designed using the bilinear transform.

CALL:        [a,b] = cz2iir(z, fs)

REMARK:      This is just a helper function. This function should only be
              called from class functions.

INPUTS:      z - zero object
              fs - the sample rate for the filter

VERSION:     $Id: cz2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:     03-04-2007 Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the
0002 % bilinear transform.
0003 %
0004 % DESCRIPTION: CZ2IIR return a,b IIR filter coefficients for a complex zero
0005 %               designed using the bilinear transform.
0006 %
0007 % CALL:        [a,b] = cz2iir(z, fs)
0008 %
0009 % REMARK:      This is just a helper function. This function should only be
0010 %               called from class functions.
0011 %
0012 % INPUTS:      z - zero object
0013 %               fs - the sample rate for the filter
0014 %
```

```
0015 % VERSION:      $Id: cz2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:       03-04-2007 Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = cz2iir(varargin)
0023
0024     %% Check if this is a call for parameters
0025     if utils.helper.isinfocall(varargin{:})
0026         varargout{1} = getInfo(varargin{3});
0027         return
0028     end
0029
0030     z = varargin{1};
0031     fs = varargin{2};
0032
0033     f0 = z.f;
0034     q = z.q;
0035
0036     w0 = f0*2*pi;
0037     w02 = w0^2;
0038
0039     a(1) = (-q*w02/2 - 2*q*fs*fs - w0*fs) / (q*w02);
0040     a(2) = (-w02+4*fs*fs) / w02;
0041     a(3) = (-q*w02/2 - 2*q*fs*fs + w0*fs) / (q*w02);
0042
0043     b(1) = 1;
0044     b(2) = -2;
0045     b(3) = -1;
0046
0047     varargout{1} = a;
0048     varargout{2} = b;
0049 end
0050
0051 %%%%%%%%%%%%%%
0052 %               Local Functions
0053 %%%%%%%%%%%%%%
0054
0055 %%%%%%%%%%%%%%
0056 %
0057 % FUNCTION:      getInfo
0058 %
0059 % DESCRIPTION:   Get Info Object
0060 %
0061 % HISTORY:       11-07-07 M Hewitson
0062 %               Creation.
0063 %
0064 %%%%%%%%%%%%%%
0065
0066 function ii = getInfo(varargin)
0067     if nargin == 1 && strcmpi(varargin{1}, 'None')
0068         sets = {};
0069         pl = [];
0070     else
0071         sets = {'Default'};
0072         pl = getDefaultPlist;
0073     end
0074     % Build info object
0075     ii = minfo(mfilename, 'pz', '', utils.const.categories.internal, '$Id: cz2iir.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0076 end
0077
0078 %%%%%%%%%%%%%%
0079 %
0080 % FUNCTION:      getDefaultPlist
0081 %
0082 % DESCRIPTION:   Get Default Plist
0083 %
0084 % HISTORY:       11-07-07 M Hewitson
0085 %               Creation.
0086 %
0087 %%%%%%%%%%%%%%
0088
0089 function plo = getDefaultPlist()
0090     plo = plist();
0091 end
0092
```


rp2iir

PURPOSE ^

RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.

SYNOPSIS ^

function varargout = rp2iir(varargin)

DESCRIPTION ^

```
RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RP2IIR Return a,b coefficients for a real pole designed using
the bilinear transform.

CALL:      filt = rpole(p, fs)

REMARK:    This is just a helper function. This function should only be
called from class functions.

INPUT:     p - pole object
           fs - the sample rate for the filter

VERSION:   $Id: rp2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:   03-03-07 M Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RP2IIR Return a,b coefficients for a real pole designed using
0005 % the bilinear transform.
0006 %
0007 % CALL:      filt = rpole(p, fs)
0008 %
0009 % REMARK:    This is just a helper function. This function should only be
0010 % called from class functions.
0011 %
0012 % INPUT:     p - pole object
0013 %           fs - the sample rate for the filter
0014 %
0015 % VERSION:   $Id: rp2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0016 %
```

```
0017 % HISTORY:      03-03-07 M Hewitson
0018 %              Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = rp2iir(varargin)
0023
0024     %% Check if this is a call for parameters
0025     if utils.helper.isinfocall(varargin{:})
0026         varargout{1} = getInfo(varargin{3});
0027         return
0028     end
0029
0030     p = varargin{1};
0031     fs = varargin{2};
0032
0033     f0 = p.f;
0034     w0 = f0*2*pi;
0035     a(1) = w0 / (2*fs + w0);
0036     a(2) = a(1);
0037     b(1) = 1;
0038     b(2) = (w0-2*fs) / (w0+2*fs);
0039
0040     varargout{1} = a;
0041     varargout{2} = b;
0042 end
0043
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                               Local Functions                               %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %              Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'pz', '', utils.const.categories.internal, '$Id: rp2iir.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0069 end
0070
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %              Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081
0082 function plo = getDefaultPlist()
0083     plo = plist();
0084 end
0085
```

rz2iir

PURPOSE ^

RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.

SYNOPSIS ^

function varargout = rz2iir(varargin)

DESCRIPTION ^

RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.
DESCRIPTION: RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.
CALL: [a,b] = rz2iir(z, fs)
REMARK: This is just a helper function. This function should only be called from class functions.
INPUT: z - zero object
fs - the sample rate for the filter
VERSION: \$Id: rz2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp \$
HISTORY: 18-02-2003 Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 pz PZ is the ltpda class that provides a common definition of poles and zeros.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear
0002 % transform.
0003 %
0004 % DESCRIPTION: RZ2IIR Return a,b IIR filter coefficients for a real zero
0005 % designed using the bilinear transform.
0006 %
0007 % CALL: [a,b] = rz2iir(z, fs)
0008 %
0009 % REMARK: This is just a helper function. This function should only be
0010 % called from class functions.
0011 %
0012 % INPUT: z - zero object
0013 % fs - the sample rate for the filter
0014 %
```

```
0015 % VERSION:      $Id: rz2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      18-02-2003 Hewitson
0018 %              Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022 function varargout = rz2iir(varargin)
0023
0024     %% Check if this is a call for parameters
0025     if utils.helper.isinfocall(varargin{:})
0026         varargout{1} = getInfo(varargin{3});
0027         return
0028     end
0029
0030     z = varargin{1};
0031     fs = varargin{2};
0032
0033     f0 = z.f;
0034     w0 = f0*2*pi;
0035
0036     a(1) = (2*fs + w0) / w0;
0037     a(2) = (-2*fs + w0) / w0;
0038
0039     b(1) = 1;
0040     b(2) = 1;
0041
0042     varargout{1} = a;
0043     varargout{2} = b;
0044 end
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0047 %                               Local Functions
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 % FUNCTION:      getInfo
0053 %
0054 % DESCRIPTION:   Get Info Object
0055 %
0056 % HISTORY:      11-07-07 M Hewitson
0057 %              Creation.
0058 %
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl = [];
0065     else
0066         sets = {'Default'};
0067         pl = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'pz', '', utils.const.categories.internal, '$Id: rz2iir.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0071 end
0072
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getDefaultPlist
0076 %
0077 % DESCRIPTION:   Get Default Plist
0078 %
0079 % HISTORY:      11-07-07 M Hewitson
0080 %              Creation.
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 function plo = getDefaultPlist()
0085     plo = plist();
0086 end
0087
```

char

PURPOSE ^

CHAR convert a pz object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a pz object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: CHAR convert a pz object into a string.
CALL:      string = char(obj)
M-FILE INFO: Get information about this methods by calling
              >> pz.getInfo('char')
              Get information about a specified set-plist by calling:
              >> pz.getInfo('char', 'set')
VERSION:    $Id: char.m,v 1.5 2008/09/04 15:29:31 ingo Exp $
HISTORY:    04-06-2008 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % CHAR convert a pz object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a pz object into a string.
0005 %
0006 % CALL:      string = char(obj)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> pz.getInfo('char')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> pz.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.5 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:    04-06-2008 M Hewitson
0017 %              Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
```

```

0021 function varargout = char(varargin)
0022
0023 % Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'pz');
0030
0031 pstr = '';
0032 for ii = 1:numel(objs)
0033     pp = objs(ii);
0034     % f and Q
0035     pstr = [pstr sprintf('(f=%g Hz,Q=%g), ', pp.f, pp.q)];
0036 end
0037
0038 %% Prepare output
0039 varargout{1} = pstr(1:end-2);
0040 end
0041
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043 %                                     Local Functions
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 % FUNCTION:      getInfo
0049 %
0050 % DESCRIPTION:   Get Info Object
0051 %
0052 % HISTORY:       11-07-07 M Hewitson
0053 %                 Creation.
0054 %
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056
0057 function ii = getInfo(varargin)
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pl = [];
0061     else
0062         sets = {'Default'};
0063         pl = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'pz', '', utils.const.categories.output, '$Id: char.m,v 1.5
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0067 end
0068
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070 %
0071 % FUNCTION:      getDefaultPlist
0072 %
0073 % DESCRIPTION:   Get Default Plist
0074 %
0075 % HISTORY:       11-07-07 M Hewitson
0076 %                 Creation.
0077 %
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079
0080 function plo = getDefaultPlist()
0081     plo = plist();
0082 end
0083
0084

```

display

PURPOSE ^

DISPLAY display a pz object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY display a pz object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY a pz object.

CALL FOR PARAMETERS:

    obj.display('INFO') % Retrieve method information

M-FILE INFO: Get information about this methods by calling
>> pz.getInfo('display')

    Get information about a specified set-plist by calling:
>> pz.getInfo('display', 'set')

VERSION:      $Id: display.m,v 1.3 2008/09/04 15:29:31 ingo Exp $

HISTORY:      30-01-2007 M Hewitson
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY display a pz object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY a pz object.
0005 %
0006 % CALL FOR PARAMETERS:
0007 %
0008 %         obj.display('INFO') % Retrieve method information
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 % >> pz.getInfo('display')
0012 %
0013 %         Get information about a specified set-plist by calling:
0014 % >> pz.getInfo('display', 'set')
0015 %
0016 % VERSION:      $Id: display.m,v 1.3 2008/09/04 15:29:31 ingo Exp $
0017 %
0018 % HISTORY:      30-01-2007 M Hewitson
```

```
0019 %                               Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = display(varargin)
0024 %
0025 % Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 % Get minfo objects
0032 objs = utils.helper.collect_objects(varargin(:), 'pz');
0033 %
0034 % get display text
0035 txt = utils.helper.objdisp(objs);
0036 %
0037 % display the objects
0038 if nargout > 0
0039     varargout{1} = txt;
0040 elseif nargout == 0;
0041     for j=1: numel(txt)
0042         disp(txt{j});
0043     end
0044 end
0045 end
0046 %
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048 %                               Local Functions                               %
0049 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0050 %
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052 %
0053 % FUNCTION:      getInfo
0054 %
0055 % DESCRIPTION:   Get Info Object
0056 %
0057 % HISTORY:       11-07-07 M Hewitson
0058 %                Creation.
0059 %
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061 %
0062 function ii = getInfo(varargin)
0063     if nargin == 1 && strcmpi(varargin{1}, 'None')
0064         sets = {};
0065         pl = [];
0066     else
0067         sets = {'Default'};
0068         pl = getDefaultPlist;
0069     end
0070     % Build info object
0071     ii = minfo(mfilename, 'pz', '', utils.const.categories.output, '$Id: display.m,v 1.3
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0072 end
0073 %
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075 %
0076 % FUNCTION:      getDefaultPlist
0077 %
0078 % DESCRIPTION:   Get Default Plist
0079 %
0080 % HISTORY:       11-07-07 M Hewitson
0081 %                Creation.
0082 %
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084 %
0085 function plo = getDefaultPlist()
0086     plo = plist();
0087 end
0088
```


string

PURPOSE ^

STRING writes a command string that can be used to recreate the input pz object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input pz object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate
             the input pz object.

CALL:       cmd = string(pz)

M-FILE INFO: Get information about this methods by calling
              >> pz.getInfo('string')

              Get information about a specified set-plist by calling:
              >> pz.getInfo('string', 'set')

VERSION:    $Id: string.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:    02-04-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input pz object.

This function is called by:

-  [pz](#) PZ is the Itpda class that provides a common definition of poles and zeros.
-  [string](#) STRING writes a command string that can be used to recreate the input pz object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input pz object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate
0005 %               the input pz object.
0006 %
0007 % CALL:       cmd = string(pz)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %               >> pz.getInfo('string')
0011 %
```

```
0012 %           Get information about a specified set-plist by calling:
0013 %           >> pz.getInfo('string', 'set')
0014 %
0015 % VERSION:      $Id: string.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0016 %
0017 % HISTORY:      02-04-2007 M Hewitson
0018 %                Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = string(varargin)
0023
0024 % Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 p = utils.helper.collect_objects(varargin(:), 'pz');
0031
0032 cmd = '';
0033 for j=1: numel(p)
0034     pl = plist('f', p(j).f, 'q', p(j).q);
0035
0036     cmd = [cmd 'pz(' string(pl) ') '];
0037 end
0038
0039 %%% Wrap the command only with brackets if the there are more than one object
0040 if numel(p) > 1
0041     cmd = ['[' cmd(1:end-1) ']'];
0042 end
0043
0044 % set output
0045 varargout{1} = cmd;
0046 end
0047
0048 %%%%%%%%%%%%%%
0049 %                               Local Functions                               %
0050 %%%%%%%%%%%%%%
0051
0052 %%%%%%%%%%%%%%
0053 %
0054 % FUNCTION:      getInfo
0055 %
0056 % DESCRIPTION:   Get Info Object
0057 %
0058 % HISTORY:      11-07-07 M Hewitson
0059 %                Creation.
0060 %
0061 %%%%%%%%%%%%%%
0062
0063 function ii = getInfo(varargin)
0064     if nargin == 1 && strcmpi(varargin{1}, 'None')
0065         sets = {};
0066         pl = [];
0067     else
0068         sets = {'Default'};
0069         pl = getDefaultPlist;
0070     end
0071     % Build info object
0072     ii = minfo(mfilename, 'pz', '', utils.const.categories.output, '$Id: string.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0073 end
0074
0075 %%%%%%%%%%%%%%
0076 %
0077 % FUNCTION:      getDefaultPlist
0078 %
0079 % DESCRIPTION:   Get Default Plist
0080 %
0081 % HISTORY:      11-07-07 M Hewitson
0082 %                Creation.
0083 %
0084 %%%%%%%%%%%%%%
0085
0086 function plo = getDefaultPlist()
0087     plo = plist();
0088 end
0089
```

resp

PURPOSE ^

RESP returns the complex response of the pz object.

SYNOPSIS ^

function [f,r] = resp(varargin)

DESCRIPTION ^

```
RESP returns the complex response of the pz object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RESP returns the complex response of the pz object. The
              response is computed assuming that object represents a pole.
              If the object represents a zero, just take the inverse of
              the returned response: 1./r.

CALL:        [f,r] = resp(p, f);           % compute response for vector f
              [f,r] = resp(p, f1, f2, nf); % compute response from f1 to f2
                                   % in nf steps.
              [f,r] = resp(p, f1, f2, nf, scale); % compute response from f1 to f2
                                   % in nf steps using scale ['lin' or
'log'].
              [f,r] = resp(p);             % compute response

REMARK:       This is just a helper function. This function should only be
              called from class functions.

M-FILE INFO:  Get information about this methods by calling
              >> pz.getInfo('resp')

              Get information about a specified set-plist by calling:
              >> pz.getInfo('resp', 'set')

VERSION:      $Id: resp.m,v 1.5 2008/09/04 15:29:31 ingo Exp $

HISTORY:      03-04-2007 Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of pz objects depending of the second input

This function is called by:

 [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % RESP returns the complex response of the pz object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0003 %
0004 % DESCRIPTION: RESP returns the complex response of the pz object. The
0005 % response is computed assuming that object represents a pole.
0006 % If the object represents a zero, just take the inverse of
0007 % the returned response: 1./r.
0008 %
0009 % CALL:      [f,r] = resp(p, f);           % compute response for vector f
0010 %           [f,r] = resp(p, f1, f2, nf); % compute response from f1 to f2
0011 %                                     % in nf steps.
0012 %           [f,r] = resp(p, f1, f2, nf, scale); % compute response from f1 to f2
0013 %                                     % in nf steps using scale ['lin' or
0014 % 'log'].
0015 %           [f,r] = resp(p);               % compute response
0016 % REMARK:    This is just a helper function. This function should only be
0017 %            called from class functions.
0018 %
0019 % M-FILE INFO: Get information about this methods by calling
0020 %              >> pz.getInfo('resp')
0021 %
0022 %            Get information about a specified set-plist by calling:
0023 %              >> pz.getInfo('resp', 'set')
0024 %
0025 % VERSION:    $Id: resp.m,v 1.5 2008/09/04 15:29:31 ingo Exp $
0026 %
0027 % HISTORY:    03-04-2007 Hewitson
0028 %              Creation
0029 %
0030 %%%%%%%%%%%
0031
0032 function [f,r] = resp(varargin)
0033
0034 % Check if this is a call for parameters
0035 if utils.helper.isinfocall(varargin{:})
0036     f = getInfo(varargin{3});
0037     return
0038 end
0039
0040 %%% Input objects checks
0041 if nargin < 1
0042     error('### incorrect number of inputs.')
0043 end
0044
0045 %%% look at inputs
0046 p = varargin{1};
0047 if ~isa(p, 'pz')
0048     error('### first argument should be a pz object.');
```

```
0088         f = logspace(log10(f1), log10(f2), nf);
0089     end
0090 end
0091
0092
0093 %%% Now compute the response
0094 if Q>=0.5
0095     re = 1 - (f.^2./f0^2);
0096     im = f ./ (f0*Q);
0097     r = 1./complex(re, im);
0098 else
0099     re = 1;
0100     im = f./f0;
0101     r = 1./complex(re, im);
0102 end
0103 end
0104
0105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106 %                                     Local Functions
0107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108
0109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0110 %
0111 % FUNCTION:      getInfo
0112 %
0113 % DESCRIPTION:  Get Info Object
0114 %
0115 % HISTORY:      11-07-07 M Hewitson
0116 %               Creation.
0117 %
0118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0119
0120 function ii = getInfo(varargin)
0121     if nargin == 1 && strcmpi(varargin{1}, 'None')
0122         sets = {};
0123         pl   = [];
0124     else
0125         sets = {'Default'};
0126         pl   = getDefaultPlist;
0127     end
0128     % Build info object
0129     ii = minfo(mfilename, 'pz', '', utils.const.categories.sigproc, '$Id: resp.m,v 1.5
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0130 end
0131
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133 %
0134 % FUNCTION:      getDefaultPlist
0135 %
0136 % DESCRIPTION:  Get Default Plist
0137 %
0138 % HISTORY:      11-07-07 M Hewitson
0139 %               Creation.
0140 %
0141 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0142
0143 function plo = getDefaultPlist()
0144     plo = plist();
0145 end
0146
```

minfo

PURPOSE ^

MINFO a helper class for LTPDA methods.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
MINFO a helper class for LTPDA methods.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MINFO is a helper class for LTPDA methods. It holds minformation about the
method.

>> ii = minfo(mname, mclass, mpackage, mcategory, mversion, param_sets, plists, argsmin,
argsmax)

Inputs:
  mname      - The method name
  mclass     - The method class
  mpackage   - The package (if applicable)
  mcategory  - The category of the method
  mversion   - The version of the method
  sets       - A cell array of parameter set names
  plists     - An array of parameter lists
  argsmin    - Minimum number of input arguments [0 is unbounded]
  argsmax    - Maximum number of input arguments [0 is unbounded]

Outputs:
  ii - an minfo object

M Hewitson 27-05-08

$Id: minfo.m,v 1.15 2008/09/03 16:34:49 hewitson Exp $
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert an minfo object into a string.
-  [copy](#) COPY Make copy of minfo objects depending of the second input
-  [display](#) DISPLAY display an minfo object.
-  [minfo](#) MINFO a helper class for LTPDA methods.
-  [setMversion](#) SETMVERSION Set the property 'mversion'.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [char](#) CHAR convert an minfo object into a string.
-  [copy](#) COPY Make copy of minfo objects depending of the second input
-  [display](#) DISPLAY display an minfo object.
-  [minfo](#) MINFO a helper class for LTPDA methods.
-  [setMversion](#) SETMVERSION Set the property 'mversion'.

SUBFUNCTIONS ^

-  function ii = minfo(varargin)
-  function out = VEROUT()
-  function ii = getInfo(varargin)
-  function out = SETS()
-  function out = getDefaultPlist(set)

SOURCE CODE ^

```
0001 % MINFO a helper class for LTPDA methods.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % MINFO is a helper class for LTPDA methods. It holds minformation about the
0005 % method.
0006 %
0007 % >> ii = minfo(mname, mclass, mpackage, mcategory, mversion, param_sets, plists,
0008 % argsmin, argsmax)
0009 %
0010 % Inputs:
0011 % mname - The method name
0012 % mclass - The method class
0013 % mpackage - The package (if applicable)
0014 % mcategory - The category of the method
0015 % mversion - The version of the method
0016 % sets - A cell array of parameter set names
0017 % plists - An array of parameter lists
0018 % argsmin - Minimum number of input arguments [0 is unbounded]
0019 % argsmax - Maximum number of input arguments [0 is unbounded]
0020 %
0021 % Outputs:
0022 % ii - an minfo object
0023 %
0024 % M Hewitson 27-05-08
0025 %
0026 % $Id: minfo.m,v 1.15 2008/09/03 16:34:49 hewitson Exp $
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 classdef minfo < ltpda_nuo
0031 %
0032 %----- Private read-only Properties -----
0033 %----- Private read-only Properties -----
0034 %----- Private read-only Properties -----
0035 properties (SetAccess = private)
0036 mname = '';
0037 mclass = '';
0038 mpackage = '';
0039 mcategory = '';
0040 mversion = '';
0041 sets = {};
0042 plists = [];
0043 argsmin = 1;
0044 argsmax = 0;
0045 end
0046
0047 properties (SetAccess = protected)
0048 version = '$Id: minfo.m,v 1.15 2008/09/03 16:34:49 hewitson Exp $';
0049 end
0050
0051 %----- Declaration of public methods -----
0052 %----- Declaration of public methods -----
0053 %----- Declaration of public methods -----
0054 methods
0055 %
0056 %----- Property rules -----
0057 %----- Property rules -----
0058 %----- Property rules -----
0059 %
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061 % Constructor
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063 function ii = minfo(varargin)
0064 %
0065 %%% Call superclass
0066 ii = ii@ltpda_nuo(varargin{:});
0067 %
0068 %%% Set default values %%%
```



```

0069
0070 switch nargin
0071     case 0
0072         case 1
0073
0074             if isstruct(varargin{1})
0075                 %%%%%%%%%%% ii = minfo(structure) %%%%%%%%%%%
0076                 %%%%%%%%%%% necessary for readxml %%%%%%%%%%%
0077
0078                 %%% Set properties which are declared in this class
0079                 ii.mname = varargin{1}.mname;
0080                 ii.mclass = varargin{1}.mclass;
0081                 ii.mpackage = varargin{1}.mpackage;
0082                 ii.mcategory = varargin{1}.mcategory;
0083                 ii.mversion = varargin{1}.mversion;
0084                 ii.sets = varargin{1}.sets;
0085                 ii.plists = varargin{1}.plist;
0086                 ii.argsmin = varargin{1}.argsmin;
0087                 ii.argsmax = varargin{1}.argsmax;
0088                 ii.version = varargin{1}.version;
0089
0090             elseif isa(varargin{1}, 'minfo')
0091                 ii = copy(varargin{1}, 1);
0092             else
0093                 ii.mname = varargin{1};
0094             end
0095
0096         case 2
0097             ii.mname = varargin{1};
0098             ii.mclass = varargin{2};
0099         case 3
0100             ii.mname = varargin{1};
0101             ii.mclass = varargin{2};
0102             ii.mpackage = varargin{3};
0103         case 4
0104             ii.mname = varargin{1};
0105             ii.mclass = varargin{2};
0106             ii.mpackage = varargin{3};
0107             ii.mcategory = varargin{4};
0108         case 5
0109             ii.mname = varargin{1};
0110             ii.mclass = varargin{2};
0111             ii.mpackage = varargin{3};
0112             ii.mcategory = varargin{4};
0113             ii.mversion = varargin{5};
0114         case 6
0115             ii.mname = varargin{1};
0116             ii.mclass = varargin{2};
0117             ii.mpackage = varargin{3};
0118             ii.mcategory = varargin{4};
0119             ii.mversion = varargin{5};
0120             ii.sets = varargin{6};
0121         case 7
0122             ii.mname = varargin{1};
0123             ii.mclass = varargin{2};
0124             ii.mpackage = varargin{3};
0125             ii.mcategory = varargin{4};
0126             ii.mversion = varargin{5};
0127             ii.sets = varargin{6};
0128             % add plist until we have at least as many as set
0129             % descriptions
0130             pls = varargin{7};
0131             while numel(pls) < numel(ii.sets)
0132                 pls = [pls plist];
0133             end
0134             ii.plist = pls;
0135             % Check we have one set description per plist
0136             if numel(ii.plist) ~= numel(ii.sets)
0137                 sets = ii.sets;
0138                 while numel(sets) < numel(ii.plist)
0139                     sets = [sets {' '}];
0140                 end
0141                 ii.sets = sets;
0142             end
0143         case 8
0144             ii.mname = varargin{1};
0145             ii.mclass = varargin{2};
0146             ii.mpackage = varargin{3};
0147             ii.mcategory = varargin{4};
0148             ii.mversion = varargin{5};
0149             ii.sets = varargin{6};
0150             % add plist until we have at least as many as set
0151             % descriptions
0152             pls = varargin{7};
0153             while numel(pls) < numel(ii.sets)
0154                 pls = [pls plist];

```



```

0155         end
0156         ii.plists = pls;
0157         % Check we have one set description per plist
0158         if numel(ii.plists) ~= numel(ii.sets)
0159             sets = ii.sets;
0160             while numel(sets) < numel(ii.plists)
0161                 sets = [sets {' '}];
0162             end
0163             ii.sets = sets;
0164         end
0165         ii.argsmin = varargin{8};
0166     case 9
0167         ii.mname = varargin{1};
0168         ii.mclass = varargin{2};
0169         ii.mpackage = varargin{3};
0170         ii.mcategory = varargin{4};
0171         ii.mversion = varargin{5};
0172         ii.sets = varargin{6};
0173         % add plists until we have at least as many as set
0174         % descriptions
0175         pls = varargin{7};
0176         while numel(pls) < numel(ii.sets)
0177             pls = [pls plist];
0178         end
0179         ii.plists = pls;
0180         % Check we have one set description per plist
0181         if numel(ii.plists) ~= numel(ii.sets)
0182             sets = ii.sets;
0183             while numel(sets) < numel(ii.plists)
0184                 sets = [sets {' '}];
0185             end
0186             ii.sets = sets;
0187         end
0188         ii.argsmin = varargin{8};
0189         ii.argsmax = varargin{9};
0190     otherwise
0191         error('### Unknown number of constructor arguments');
0192     end
0193 end % End of constructor
0194 end % End public methods
0195
0196 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0197 %                                     Methods (public)                                     %
0198 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0199
0200 methods
0201     varargout = char(varargin)
0202     varargout = copy(varargin)
0203     varargout = display(varargin)
0204     varargout = setMversion(varargin)
0205 end
0206
0207 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0208 %                                     Methods (protected)                                     %
0209 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0210
0211 methods (Access = protected)
0212 end
0213
0214 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0215 %                                     Methods (Static, Public)                                     %
0216 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0217
0218 methods (Static = true)
0219
0220     varargout = update_struct(varargin);
0221
0222     function out = VEROUT()
0223         out = '$Id: minfo.m,v 1.15 2008/09/03 16:34:49 hewitson Exp $';
0224     end
0225
0226     function ii = getInfo(varargin)
0227         ii = utils.helper.generic_getInfo(varargin{:}, 'minfo');
0228     end
0229
0230     function out = SETS()
0231         out = {'Default'};
0232     end
0233
0234     function out = getDefaultPlist(set)
0235         switch set
0236         case 'Default'
0237             out = plist();
0238         otherwise
0239             out = plist();
0240         end

```

```
0241         end
0242
0243     end
0244
0245     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0246     %                               Methods (Static, Private)                               %
0247     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0248
0249     methods (Static = true, Access = private)
0250     end
0251
0252 end
0253
0254 % END
```

copy

PURPOSE ^

COPY Make copy of minfo objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of minfo objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of minfo objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> minfo.getInfo('copy')

VERSION:    $Id: copy.m,v 1.6 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [minfo](#) MINFO a helper class for LTPDA methods.

This function is called by:

-  [minfo](#) MINFO a helper class for LTPDA methods.
-  [setMversion](#) SETMVERSION Set the property 'mversion'.

SOURCE CODE ^

```
0001 % COPY Make copy of minfo objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of minfo objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> minfo.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.6 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('minfo');
0022         s = size(old);
0023         obj(s(1),s(2)) = minfo;
```

```
0024     for kk=1:numel(old)
0025         utils.helper.msg(utils.const.msg.OPROC2, 'copying minfo');
0026         for jj=1:length(ps)
0027             obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028         end
0029         % reset on copy
0030         obj(kk).version = minfo.VEROUT;
0031     end
0032 else
0033     obj = old;
0034 end
0035 varargout{1} = obj;
0036 end
```

setMversion

PURPOSE ^

SETMVERSION Set the property 'mversion'.

SYNOPSIS ^

function varargout = setMversion(varargin)

DESCRIPTION ^

```
SETMVERSION Set the property 'mversion'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'mversion'.

CALL:
    obj.setMversion([1 2 3]);
    obj = obj.setMversion([1 2 3]); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> minfo.getInfo('setMversion')

    Get information about a specified set-plist by calling:
    >> minfo.getInfo('setMversion', 'None')

VERSION:
    $Id: setMversion.m,v 1.3 2008/09/04 15:33:01 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of minfo objects depending of the second input
-  [minfo](#) MINFO a helper class for LTPDA methods.

This function is called by:

-  [minfo](#) MINFO a helper class for LTPDA methods.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETMVERSION Set the property 'mversion'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'mversion'.
0005 %
0006 % CALL:
0007 %     obj.setMversion([1 2 3]);
0008 %     obj = obj.setMversion([1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
```

```
0010 %
0011 % M-FILE INFO: Get information about this methods by calling
0012 % >> minfo.getInfo('setMversion')
0013 %
0014 % Get information about a specified set-plist by calling:
0015 % >> minfo.getInfo('setMversion', 'None')
0016 %
0017 % VERSION: $Id: setMversion.m,v 1.3 2008/09/04 15:33:01 ingo Exp $
0018 %
0019 % HISTORY: 27-05-2008 Diepholz
0020 % Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = setMversion(varargin)
0025
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031
0032 obj = varargin{1};
0033 val = varargin{2};
0034
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037
0038 %%% set 'mversion'
0039 obj.mversion = val;
0040
0041 varargout{1} = obj;
0042 end
0043
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 % Local Functions %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION: getInfo
0051 %
0052 % DESCRIPTION: Get Info Object
0053 %
0054 % HISTORY: 11-07-07 M Hewitson
0055 % Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059 function ii = getInfo(varargin)
0060 if nargin == 1 && strcmpi(varargin{1}, 'None')
0061     sets = {};
0062     pl = [];
0063 else
0064     sets = {'Default'};
0065     pl = getDefaultPlist;
0066 end
0067 % Build info object
0068 ii = minfo(mfilename, 'minfo', '', utils.const.categories.internal, '$Id:
setMversion.m,v 1.3 2008/09/04 15:33:01 ingo Exp $', sets, pl);
0069 end
0070
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION: getDefaultPlist
0074 %
0075 % DESCRIPTION: Get Default Plist
0076 %
0077 % HISTORY: 11-07-07 M Hewitson
0078 % Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081
0082 function plo = getDefaultPlist()
0083 plo = plist('mversion', '');
0084 end
0085
```

char

PURPOSE ^

CHAR convert an minfo object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert an minfo object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: CHAR convert an minfo object into a string.
CALL:      string = char(obj)
CALL FOR PARAMETERS:
            obj.char('INFO') % Struct with all function parameters
VERSION:   $Id: char.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
HISTORY:   04-06-2008 M Hewitson
            Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [minfo](#) MINFO a helper class for LTPDA methods.

This function is called by:

 [minfo](#) MINFO a helper class for LTPDA methods.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert an minfo object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert an minfo object into a string.
0005 %
0006 % CALL:      string = char(obj)
0007 %
0008 % CALL FOR PARAMETERS:
0009 %
0010 %      obj.char('INFO') % Struct with all function parameters
0011 %
0012 % VERSION:   $Id: char.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0013 %
0014 % HISTORY:   04-06-2008 M Hewitson
0015 %            Creation
0016 %
```

```
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 function varargout = char(varargin)
0020
0021 %%% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 objs = utils.helper.collect_objects(varargin(:), 'minfo');
0028
0029 pstr = '';
0030 for ii = 1:numel(objs)
0031
0032     pp = objs(ii);
0033     % method class
0034     pstr = [pstr sprintf('%s/', pp.mclass)];
0035     % method name
0036     pstr = [pstr sprintf('%s', pp.mname)];
0037     % method category
0038     pstr = [pstr sprintf(', %s', pp.mcategory)];
0039     % method sets
0040     if ~isempty(pp.sets)
0041         pstr = [pstr sprintf(', %s', utils.prog.cell2str(pp.sets))];
0042     end
0043 end
0044
0045 %%% Prepare output
0046 varargout{1} = pstr;
0047 end
0048
0049 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0050 %                               Local Functions                               %
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054 %
0055 % FUNCTION:      getInfo
0056 %
0057 % DESCRIPTION:  Get Info Object
0058 %
0059 % HISTORY:      11-07-07 M Hewitson
0060 %               Creation.
0061 %
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063
0064 function ii = getInfo(varargin)
0065     if nargin == 1 && strcmpi(varargin{1}, 'None')
0066         sets = {};
0067         pl = [];
0068     else
0069         sets = {'Default'};
0070         pl = getDefaultPlist;
0071     end
0072     % Build info object
0073     ii = minfo(mfilename, 'minfo', '', utils.const.categories.output, '$Id: char.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0074 end
0075
0076 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077 %
0078 % FUNCTION:      getDefaultPlist
0079 %
0080 % DESCRIPTION:  Get Default Plist
0081 %
0082 % HISTORY:      11-07-07 M Hewitson
0083 %               Creation.
0084 %
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086
0087 function plo = getDefaultPlist()
0088     plo = plist();
0089 end
0090
```


display

PURPOSE ^

DISPLAY display an minfo object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY display an minfo object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY display an minfo object.

CALL FOR PARAMETERS:

            obj.display('INFO') % Retrieve method information

VERSION:    $Id: display.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:    30-01-2007 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 minfo MINFO a helper class for LTPDA methods.

This function is called by:

 minfo MINFO a helper class for LTPDA methods.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % DISPLAY display an minfo object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY display an minfo object.
0005 %
0006 % CALL FOR PARAMETERS:
0007 %
0008 %             obj.display('INFO') % Retrieve method information
0009 %
0010 % VERSION:    $Id: display.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0011 %
0012 % HISTORY:    30-01-2007 M Hewitson
0013 %             Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 function varargout = display(varargin)
0018
```

```
0019 %% Check if this is a call for parameters
0020 if utils.helper.isinfocall(varargin{:})
0021     varargin{1} = getInfo(varargin{3});
0022     return
0023 end
0024
0025 % Get minfo objects
0026 objs = utils.helper.collect_objects(varargin(:), 'minfo');
0027
0028 % get display text
0029 txt = utils.helper.objdisp(objs);
0030
0031 % display the objects
0032 if nargout > 0
0033     vararginout{1} = txt;
0034 elseif nargout == 0;
0035     for j=1:numel(txt)
0036         disp(txt{j});
0037     end
0038 end
0039 end
0040
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %                                     Local Functions                                     %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:      getInfo
0048 %
0049 % DESCRIPTION:   Get Info Object
0050 %
0051 % HISTORY:       11-07-07 M Hewitson
0052 %                 Creation.
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl    = [];
0060     else
0061         sets = {'Default'};
0062         pl    = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'minfo', '', utils.const.categories.output, '$Id: display.m,v
1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:      getDefaultPlist
0071 %
0072 % DESCRIPTION:   Get Default Plist
0073 %
0074 % HISTORY:       11-07-07 M Hewitson
0075 %                 Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082
```

history

PURPOSE ^

HISTORY History object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
HISTORY History object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: HISTORY History object class constructor.
             Create a history object.

SUPER CLASSES: ltpda_nuo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
        version      - cvs-version string.

    Protected Properties (read only)
        methodInfo   - minfo-object which is created in the called method.
        plistUsed    - plist-object which is used in the called method.
        methodInvars - variable names which are used for the called method.
        inhists      - the older history-objects
        proctime      - creation time of the history object.

    Private Properties

HISTORY METHODS:

    Defined Abstract methods:
        char          - returns one character string which represents the object
        copy          - copies an object
        display       - displays an object
        update_struct - updates a object structure to the current tbx-version

    Public methods:
        getNodes      - converts a history object to a nodes structure
        getNodes_plot - converts a history object to a nodes structure with max depth
        hist2dot       - converts a history object to a 'DOT' file suitable for
                        processing with graphviz
        hist2m         - writes a new m-file that reproduces the analysis
                        described in the history object.
        plot           - plots a history object as a tree diagram.
        string         - writes a command string that can be used to recreate
                        the input history object.

CONSTRUCTORS:

        h = history()
        h = history(proctime, minfo, plist)
        h = history(proctime, minfo, plist, in_names, in_hists)

        h = history(filename)
        h = history(structure)
        h = history(history-object)
        h = history('database', ...)

INPUTS:
    minfo:      Minfo-object which is created in the called method.
    plist:      Plist-object which is used in the called method.
    in_hist:    Older history-objects
    var_names:  Variable names which are used for the called method.

M-FILE INFO: The following call returns an minfo object that contains
              information about the history constructor:
              >> info = history.getInfo
              or  >> info = history.getInfo('history')
```

```
You can get information about class methods by calling:
>> info = history.getInfo(method)
e.g. >> info = history.getInfo('eq')

You can also restrict the sets of parameters contained in
the minfo object by calling:
>> info = history.getInfo(method, set)
e.g. >> info = history.getInfo('history', 'Default')

VERSION:      $Id: history.m,v 1.41 2008/09/08 08:30:40 hewitson Exp $

HISTORY:      19-05-2008 Diepholz
                Creation.

SEE ALSO:      ltpda_obj, ltpda_nuo, minfo, plist

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [copy](#) COPY Make copy of history objects depending of the second input
-  [display](#) DISPLAY implement terminal display for history object.
-  [getNodes](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
-  [getNodes_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
-  [hist2dot](#) HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz
-  [hist2m](#) HIST2M writes a new m-file that reproduces the analysis described in the history object.
-  [history](#) HISTORY History object class constructor.
-  [plot](#) PLOT plots a history object as a tree diagram.
-  [string](#) STRING writes a command string that can be used to recreate the input history object.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [copy](#) COPY Make copy of history objects depending of the second input
-  [getNodes](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
-  [getNodes_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
-  [history](#) HISTORY History object class constructor.

SUBFUNCTIONS ^

-  [function obj = set.methodInfo\(obj, val\)](#)
-  [function obj = set.plistUsed\(obj, val\)](#)
-  [function obj = set.methodInvars\(obj, val\)](#)
-  [function obj = set.inhists\(obj, val\)](#)
-  [function obj = history\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 % HISTORY History object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: HISTORY History object class constructor.
0005 %           Create a history object.
0006 %
0007 % SUPER CLASSES: ltpda_nuo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         version      - cvs-version string.
0013 %
0014 %     Protected Properties (read only)
0015 %         methodInfo    - minfo-object which is created in the called method.
0016 %         plistUsed     - plist-object which is used in the called method.
0017 %         methodInvars  - variable names which are used for the called method.
0018 %         inhists       - the older history-objects
0019 %         proctime      - creation time of the history object.
0020 %
0021 %     Private Properties
0022 %
0023 % HISTORY METHODS:
0024 %
0025 %     Defined Abstract methods:
0026 %         char           - returns one character string which represents the object
0027 %         copy           - copies an object
0028 %         display        - displays an object
0029 %         update_struct  - updates a object structure to the current tbx-version
0030 %
0031 %     Public methods:
0032 %         getNodes       - converts a history object to a nodes structure
0033 %         getNodes_plot  - converts a history object to a nodes structure with max
0034 %         hist2dot       - converts a history object to a 'DOT' file suitable for
0035 %                         processing with graphviz
0036 %         hist2m         - writes a new m-file that reproduces the analysis
0037 %                         described in the history object.
0038 %         plot           - plots a history object as a tree diagram.
0039 %         string         - writes a command string that can be used to recreate
0040 %                         the input history object.
0041 %
0042 % CONSTRUCTORS:
0043 %
0044 %         h = history()
0045 %         h = history(proctime, minfo, plist)
0046 %         h = history(proctime, minfo, plist, in_names, in_hists)
0047 %
0048 %         h = history(filename)
0049 %         h = history(structure)
0050 %         h = history(history-object)
0051 %         h = history('database', ...)
0052 %
0053 % INPUTS:      minfo:      Minfo-object which is created in the called method.
0054 %              plist:      Plist-object which is used in the called method.
0055 %              in_hist:    Older history-objects
0056 %              var_names:  Variable names which are used for the called method.
0057 %
0058 % M-FILE INFO: The following call returns an minfo object that contains
0059 %              information about the history constructor:
0060 %              >> info = history.getInfo
0061 %              or   >> info = history.getInfo('history')
0062 %
0063 %              You can get information about class methods by calling:
0064 %              >> info = history.getInfo(method)
0065 %              e.g. >> info = history.getInfo('eq')
0066 %
0067 %              You can also restrict the sets of parameters contained in
0068 %              the minfo object by calling:
0069 %              >> info = history.getInfo(method, set)
0070 %              e.g. >> info = history.getInfo('history', 'Default')
0071 %
0072 % VERSION:     $Id: history.m,v 1.41 2008/09/08 08:30:40 hewitson Exp $
0073 %
0074 % HISTORY:     19-05-2008 Diepholz
0075 %              Creation.
0076 %
0077 % SEE ALSO:    ltpda_obj, ltpda_nuo, minfo, plist
0078 %
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080
0081 classdef (Sealed = true) history < ltpda_nuo
0082
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084 %                                Property definition                                %
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086
0087 %----- Public (read/write) Properties -----
0088 properties

```

```

0089 end
0090
0091 %----- Protected read-only Properties -----
0092 properties (SetAccess = protected)
0093     methodInfo = [];
0094     plistUsed = [];
0095     methodInvars = [];
0096     inhists = [];
0097     proctime = [];
0098     version = '$Id: history.m,v 1.41 2008/09/08 08:30:40 hewitson Exp $';
0099 end
0100
0101 %----- Private Properties -----
0102 properties (GetAccess = public, SetAccess = private)
0103 end
0104
0105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106 %                               Check property setting                               %
0107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108
0109 methods
0110     function obj = set.methodInfo(obj, val)
0111         if ~(isa(val, 'minfo') || isempty(val))
0112             error('### The value for the property 'methodInfo'\n### must be a minfo-
object or empty but\n### it is from the class %s', class(val));
0113         end
0114         obj.methodInfo = val;
0115     end
0116     function obj = set.plistUsed(obj, val)
0117         if ~(isa(val, 'plist') || isempty(val))
0118             error('### The value for the property 'plistUsed'\n### must be a plist-object
or empty but\n### it is from the class %s', class(val));
0119         end
0120         obj.plistUsed = val;
0121     end
0122     function obj = set.methodInvars(obj, val)
0123         if ~iscell(val)
0124             error('### The value for the property 'methodInvars'\n### must be a cell-
array but\n### it is from the class %s', class(val));
0125         end
0126         obj.methodInvars = val;
0127     end
0128     function obj = set.inhists(obj, val)
0129         if ~(isa(val, 'history') || isempty(val))
0130             error('### The value for the property 'inhists'\n### must be a history-object
or empty but\n### it is from the class %s', class(val));
0131         end
0132         obj.inhists = val;
0133     end
0134 end
0135
0136 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0137 %                               Constructor                               %
0138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0139
0140 methods
0141
0142     function obj = history(varargin)
0143
0144         %% Call superclass
0145         obj = obj@ltpda_nuo(varargin{:});
0146
0147         %%%%%%%%% Set default values %%%%%%%%%
0148
0149         %%%%%%%%% no input %%%%%%%%%
0150         %%%%%%%%% no input %%%%%%%%%
0151         %%%%%%%%% no input %%%%%%%%%
0152         %%%%%%%%% no input %%%%%%%%%
0153
0154         if nargin == 0
0155
0156             % Do nothing
0157
0158             %%%%%%%%% one input %%%%%%%%%
0159             %%%%%%%%% one input %%%%%%%%%
0160             %%%%%%%%% one input %%%%%%%%%
0161         elseif nargin == 1
0162
0163             %%%%%%%%% From File %%%%%%%%%
0164             if ischar(varargin{1})
0165
0166                 filename = varargin{1};
0167                 [path, name, ext, vers] = fileparts(filename);
0168                 switch ext
0169                     case '.mat'
0170                         obj = load(filename);

```

```

0171         obj = obj.a;
0172         case '.xml'
0173             root_node = xmlread(filename);
0174             obj = utils.helper.xmlread(root_node, 'history');
0175         otherwise
0176             error('### Unknown file type.');
```

end

```

0178
0179         %%%%%%%%%% h = history(struct) %%%%%%%%%%
0180     elseif isstruct(varargin{1})
0181
0182         hstruct = varargin{1};
0183
0184         obj.methodInfo = utils.helper.struct2obj(hstruct.methodInfo, 'minfo');
0185         obj.plistUsed = utils.helper.struct2obj(hstruct.plistUsed, 'plist');
0186         obj.methodInvars = hstruct.methodInvars;
0187         obj.inhists = utils.helper.struct2obj(hstruct.inhists, 'history');
0188         obj.proctime = hstruct.proctime;
0189         obj.version = hstruct.version;
0190
0191         %%%%%%%%%% h = history(history) %%%%%%%%%%
0192     elseif isa(varargin{1}, 'history')
0193         obj = copy(varargin{1}, 1);
0194
0195         %%%%%%%%%% h = history(plist) %%%%%%%%%%
0196     elseif isa(varargin{1}, 'plist')
0197         %%% is the plist is empty then return an empty history object
0198         if nparams(varargin{1}) == 0
0199             obj.plistUsed = varargin{1};
0200         else
0201             error('### Unknown history constructor method.');
```

end

```

0203
0204     else
0205         error('### Unknown history constructor method.');
```

end

```

0207
0208     %%%%%%%%%% two input %%%%%%%%%%
0209     %%%%%%%%%% two input %%%%%%%%%%
0210     %%%%%%%%%% two input %%%%%%%%%%
0211
0212     elseif nargin == 2
0213         %%%%%%%%%% Database
0214         %%%%%%%%%% h = history('database', ...) %%%%%%%%%%
0215         if isa(varargin{1}, 'database')
0216             obj = retrieve(varargin{1}, varargin{2:end});
0217
0218         %%%%%%%%%% h = history(proctime, minfo) %%%%%%%%%%
0219     else
0220         error('### Unknown constructor method for two inputs.');
```

end

```

0222
0223     %%%%%%%%%% three inputs %%%%%%%%%%
0224     %%%%%%%%%% three inputs %%%%%%%%%%
0225     %%%%%%%%%% three inputs %%%%%%%%%%
0226
0227     %%%%%%%%%% h = history(proctime, minfo, plist) %%%%%%%%%%
0228     elseif nargin == 3
0229
0230         %%% Use default values
0231         if ~isempty(varargin{1})
0232             obj.proctime = varargin{1};
0233         end
0234         if ~isempty(varargin{2})
0235             obj.methodInfo = varargin{2};
0236         end
0237         if ~isempty(varargin{3})
0238             obj.plistUsed = varargin{3};
0239         end
0240
0241         %%%%%%%%%% five inputs %%%%%%%%%%
0242         %%%%%%%%%% five inputs %%%%%%%%%%
0243         %%%%%%%%%% five inputs %%%%%%%%%%
0244
0245         %%%%%%%%%% h = history(proctime, minfo, plist, in_names, in_hists) %%%%%%%%%%
0246     elseif nargin == 5
0247
0248         %%% Use default values
0249         if ~isempty(varargin{1})
0250             obj.proctime = varargin{1};
0251         end
0252         if ~isempty(varargin{2})
0253             obj.methodInfo = varargin{2};
0254         end
0255         if ~isempty(varargin{3})
0256             obj.plistUsed = varargin{3};
```



```

0257     end
0258     if ~isempty(varargin{4})
0259         obj.methodInvars = varargin{4};
0260     end
0261     if ~isempty(varargin{5})
0262         obj.inhists      = varargin{5};
0263     end
0264
0265     else
0266         error('### Unknown number of constructor arguments.');
```

0267 end

0268

0269 end

0270

0271 end

0272

0273

0274 %%

0275 %% Methods (public) %%

0276 %%

0277

0278 methods

0279 %%% Define Abstract methods

0280 varargout = char(varargin)

0281 varargout = copy(varargin)

0282 varargout = display(varargin)

0283

0284 varargout = getNodes(varargin)

0285 varargout = getNodes_plot(varargin)

0286 varargout = hist2dot(varargin)

0287 varargout = hist2m(varargin)

0288 varargout = plot(varargin)

0289 varargout = string(varargin)

0290 end

0291

0292 %%

0293 %% Methods (protected) %%

0294 %%

0295

0296 methods (Access = protected)

0297 end

0298

0299 %%

0300 %% Methods (Static, Public) %%

0301 %%

0302

0303 methods (Static = true)

0304

0305 varargout = update_struct(varargin);

0306

0307 function out = VEROUT()

0308 out = '\$Id: history.m,v 1.41 2008/09/08 08:30:40 hewitson Exp \$';

0309 end

0310

0311 function ii = getInfo(varargin)

0312 ii = utils.helper.generic_getInfo(varargin{:}, 'history');

0313 end

0314

0315 function out = SETS()

0316 out = {'Default'};

0317 end

0318

0319 function out = getDefaultPlist(set)

0320 switch set

0321 case 'Default'

0322 out = plist();

0323

0324 otherwise

0325 out = plist();

0326 end

0327 end

0328

0329 end

0330

0331 %%

0332 %% Methods (Static, Private) %%

0333 %%

0334

0335 methods (Static = true, Access = private)

0336 end

0337

0338 end

0339

0340

0341

0342

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

copy

PURPOSE ^

COPY Make copy of history objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

COPY Make copy of history objects depending of the second input

DESCRIPTION: COPY Make copy of history objects depending of the second input

CALL: objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
>> history.getInfo('copy')

VERSION: \$Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp \$

HISTORY: 11-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 [history](#) HISTORY History object class constructor.

This function is called by:

 [history](#) HISTORY History object class constructor.

SOURCE CODE ^

```
0001 % COPY Make copy of history objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of history objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> history.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('history');
0022         s = size(old);
0023         obj(s(1),s(2)) = history;
0024         for kk=1: numel(old)
0025             utils.helper.msg(utils.const.msg.OPROC2, 'copying history');
```

```
0026     for jj=1:length(ps)
0027         obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028     end
0029     % reset on copy
0030     obj(kk).version = history.VEROUT;
0031 end
0032 else
0033     obj = old;
0034 end
0035 varargout{1} = obj;
0036 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

getNodes

PURPOSE ^

GETNODES converts a history object to a nodes structure suitable for plotting as a tree.

SYNOPSIS ^

function varargout = getNodes(varargin)

DESCRIPTION ^

```
GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: GETNODES converts a history object to a nodes structure suitable
              for plotting as a tree.

CALL:        [n,a, nodes] = getNodes(hist, [], 0, 1, []);

OUTPUT:      n:      A vector of parent pointers
              a:      Number of nodes
              nodes:  Struct of the nodes

M-FILE INFO: Get information about this methods by calling
              >> history.getInfo('getNodes')

              Get information about a specified set-plist by calling:
              >> history.getInfo('getNodes', 'set')

VERSION:     $Id: getNodes.m,v 1.19 2008/09/04 15:29:30 ingo Exp $

HISTORY:     02-02-2007 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [getNodes](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
-  [history](#) HISTORY History object class constructor.

This function is called by:

-  [getNodes](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
-  [hist2dot](#) HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz
-  [hist2m](#) HIST2M writes a new m-file that reproduces the analysis described in the history object.
-  [history](#) HISTORY History object class constructor.
-  [plot](#) PLOT plots a history object as a tree diagram.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```

0001 % GETNODES converts a history object to a nodes structure suitable for plotting as a
0002 %%%
0003 %
0004 % DESCRIPTION: GETNODES converts a history object to a nodes structure suitable
0005 %               for plotting as a tree.
0006 %
0007 % CALL:        [n,a, nodes] = getNodes(hist, [], 0, 1, []);
0008 %
0009 % OUTPUT:      n:      A vector of parent pointers
0010 %              a:      Number of nodes
0011 %              nodes: Struct of the nodes
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> history.getInfo('getNodes')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> history.getInfo('getNodes', 'set')
0018 %
0019 % VERSION:     $Id: getNodes.m,v 1.19 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY:     02-02-2007 M Hewitson
0022 %               Creation
0023 %
0024 %%%
0025 %
0026 function varargout = getNodes(varargin)
0027
0028     import utils.const.*
0029
0030     %% Check if this is a call for parameters
0031     if utils.helper.isinfocall(varargin{:})
0032         varargout{1} = getInfo(varargin{3});
0033         return
0034     end
0035
0036     %% Set inputs
0037     h = varargin{1};
0038     n = varargin{2}; % list of current node values
0039     pn = varargin{3}; % current node value
0040     a = varargin{4}; % number of nodes
0041     nodes = varargin{5};
0042
0043     n = [n pn];
0044     pl = h.plistUsed;
0045     nodes(a).pn = pn;
0046     nodes(a).invars = h.methodInvars;
0047     nodes(a).n = a;
0048     nodes(a).names = h.methodInfo.mname;
0049     nodes(a).params = char(pl);
0050     nodes(a).pl = pl;
0051     nodes(a).h = history(h);
0052     utils.helper.msg(msg.OPROC1, 'scanning node %03d [%s]', a, char(nodes(a).names));
0053
0054     % set the current node value to the number of the child
0055     pn = a-1;
0056
0057     ih = h.inhists;
0058
0059     % Now decide what to do with my children
0060     if isa(ih, 'history')
0061         for i=1:length(ih)
0062             a = a + 1;
0063
0064             [n, a, nodes] = getNodes(ih(i), n, pn+1, a, nodes);
0065
0066         end
0067     end
0068
0069     % Set outputs
0070     varargout{1} = n;
0071     varargout{2} = a;
0072     varargout{3} = nodes;
0073 end
0074
0075 %%%
0076 %               Local Functions
0077 %%%
0078 %
0079 %%%
0080 %
0081 % FUNCTION:     getInfo
0082 %
0083 % DESCRIPTION:  Get Info Object
0084 %

```

```
0085 % HISTORY:      11-07-07 M Hewitson
0086 %
0087 %
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089
0090 function ii = getInfo(varargin)
0091     if nargin == 1 && strcmpi(varargin{1}, 'None')
0092         sets = {};
0093         pl   = [];
0094     else
0095         sets = {'Default'};
0096         pl   = getDefaultPlist;
0097     end
0098     % Build info object
0099     ii = minfo(mfilename, 'history', '', utils.const.categories.internal, '$Id:
getNodes.m,v 1.19 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0100 end
0101
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103 %
0104 % FUNCTION:      getDefaultPlist
0105 %
0106 % DESCRIPTION:   Get Default Plist
0107 %
0108 % HISTORY:      11-07-07 M Hewitson
0109 %
0110 %
0111 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0112
0113 function plo = getDefaultPlist()
0114     plo = plist();
0115 end
0116
```

getNodes_plot

PURPOSE ^

GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.

SYNOPSIS ^

function [varargout] = getNodes_plot(varargin)

DESCRIPTION ^

```
GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a
tree.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: GETNODES_PLOT converts a history object to a nodes structure
             suitable for plotting as a tree.

CALL:       [n,a, nodes] = getNodes_plot(hist, [], 0, 1, [], 0, 2);

OUTPUT:     n:      A vector of parent pointers
             a:      Number of nodes
             nodes:  Struct of the nodes

REMARK:     This version is equivalent to getNodes with the different that
             this function only return the nodes up to the MAX_DEPTH.

M-FILE INFO: Get information about this methods by calling
             >> history.getInfo('getNodes_plot')

             Get information about a specified set-plist by calling:
             >> history.getInfo('getNodes_plot', 'set')

VERSION:    $Id: getNodes_plot.m,v 1.9 2008/09/04 15:29:30 ingo Exp $

HISTORY:    14-06-07 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [getNodes_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
-  [history](#) HISTORY History object class constructor.

This function is called by:

-  [getNodes_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
-  [history](#) HISTORY History object class constructor.
-  [plot](#) PLOT plots a history object as a tree diagram.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```

0001 % GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as
0002 % a tree.
0003 %
0004 % DESCRIPTION: GETNODES_PLOT converts a history object to a nodes structure
0005 % suitable for plotting as a tree.
0006 %
0007 % CALL: [n,a, nodes] = getNodes_plot(hist, [], 0, 1, [], 0, 2);
0008 %
0009 % OUTPUT: n: A vector of parent pointers
0010 % a: Number of nodes
0011 % nodes: Struct of the nodes
0012 %
0013 % REMARK: This version is equivalent to getNodes with the different that
0014 % this function only return the nodes up to the MAX_DEPTH.
0015 %
0016 % M-FILE INFO: Get information about this methods by calling
0017 % >> history.getInfo('getNodes_plot')
0018 %
0019 % Get information about a specified set-plot by calling:
0020 % >> history.getInfo('getNodes_plot', 'set')
0021 %
0022 % VERSION: $Id: getNodes_plot.m,v 1.9 2008/09/04 15:29:30 ingo Exp $
0023 %
0024 % HISTORY: 14-06-07 M Hewitson
0025 % Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028 %
0029 function [varargout] = getNodes_plot(varargin)
0030
0031 %%% Check if this is a call for parameters
0032 if utils.helper.isinfocall(varargin{:})
0033     varargout{1} = getInfo(varargin{3});
0034     return
0035 end
0036
0037 %%% Set inputs
0038 h = varargin{1};
0039 n = varargin{2}; % list of current node values
0040 pn = varargin{3}; % current node value
0041 a = varargin{4}; % number of nodes
0042 nodes = varargin{5};
0043 depth = varargin{6};
0044 MAX_DEPTH = varargin{7};
0045
0046 n = [n pn];
0047 pl = h.plistUsed;
0048 nodes(a).pn = pn;
0049 nodes(a).invars = h.methodInvars;
0050 nodes(a).n = a;
0051 nodes(a).names = h.methodInfo.mname;
0052 nodes(a).params = char(pl);
0053 nodes(a).pl = pl;
0054 nodes(a).h = history(h);
0055 disp(sprintf(' -- scanning node %03d [%s]', a, char(nodes(a).names)));
0056
0057 if depth == MAX_DEPTH
0058     nodes(a).names = '...';
0059     nodes(a).params = ' ';
0060     nodes(a).invars = {};
0061 end
0062
0063 % set the current node value to the number of the child
0064 pn = a-1;
0065
0066 ih = h.inhists;
0067
0068 % Now decide what to do with my children
0069 if isa(ih, 'history')
0070     for i=1:length(ih)
0071         if depth < MAX_DEPTH
0072             a = a + 1;
0073             [n,a, nodes] = getNodes_plot(ih(i), n, pn+1, a, nodes, depth+1, MAX_DEPTH);
0074         end
0075     end
0076 end
0077
0078 % Set outputs
0079 varargout{1} = n;

```



```
0081     varargout{2} = a;
0082     varargout{3} = nodes;
0083 end
0084
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086 %                                Local Functions                                %
0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088
0089 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090 %
0091 % FUNCTION:      getInfo
0092 %
0093 % DESCRIPTION:   Get Info Object
0094 %
0095 % HISTORY:       11-07-07 M Hewitson
0096 %                Creation.
0097 %
0098 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0099
0100 function ii = getInfo(varargin)
0101     if nargin == 1 && strcmpi(varargin{1}, 'None')
0102         sets = {};
0103         pl   = [];
0104     else
0105         sets = {'Default'};
0106         pl   = getDefaultPlist;
0107     end
0108     % Build info object
0109     ii = minfo(mfilename, 'history', '', utils.const.categories.internal, '$Id:
getNodes_plot.m,v 1.9 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0110 end
0111
0112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113 %
0114 % FUNCTION:      getDefaultPlist
0115 %
0116 % DESCRIPTION:   Get Default Plist
0117 %
0118 % HISTORY:       11-07-07 M Hewitson
0119 %                Creation.
0120 %
0121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122
0123 function plo = getDefaultPlist()
0124     plo = plist();
0125 end
0126
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a param object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a history object into a string.

CALL:      string = char(hist)

M-FILE INFO: Get information about this methods by calling
              >> history.getInfo('char')

              Get information about a specified set-plist by calling:
              >> history.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.8 2008/09/04 15:29:30 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [getNode](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
-  [getNode_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
-  [hist2dot](#) HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz
-  [hist2m](#) HIST2M writes a new m-file that reproduces the analysis described in the history object.
-  [history](#) HISTORY History object class constructor.
-  [plot](#) PLOT plots a history object as a tree diagram.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a param object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a history object into a string.
0005 %
0006 % CALL:      string = char(hist)
0007 %
```

```

0008 % M-FILE INFO: Get information about this methods by calling
0009 % >> history.getInfo('char')
0010 %
0011 % Get information about a specified set-plist by calling:
0012 % >> history.getInfo('char', 'set')
0013 %
0014 % VERSION: $Id: char.m,v 1.8 2008/09/04 15:29:30 ingo Exp $
0015 %
0016 % HISTORY: 04-02-2007 M Hewitson
0017 % Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'history');
0030
0031 hstr = '';
0032
0033 for ii = 1:numel(objs)
0034     hstr = [hstr, objs(ii).methodInfo.mname, ' + '];
0035 end
0036
0037 hstr = hstr(1:end-2);
0038
0039 varargout{1} = hstr;
0040 end
0041
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043 % Local Functions %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 % FUNCTION: getInfo
0049 %
0050 % DESCRIPTION: Get Info Object
0051 %
0052 % HISTORY: 11-07-07 M Hewitson
0053 % Creation.
0054 %
0055 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056
0057 function ii = getInfo(varargin)
0058     if nargin == 1 && strcmpi(varargin{1}, 'None')
0059         sets = {};
0060         pl = [];
0061     else
0062         sets = {'Default'};
0063         pl = getDefaultPlist;
0064     end
0065     % Build info object
0066     ii = minfo(mfilename, 'history', '', utils.const.categories.output, '$Id: char.m,v 1.8
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0067 end
0068
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070 %
0071 % FUNCTION: getDefaultPlist
0072 %
0073 % DESCRIPTION: Get Default Plist
0074 %
0075 % HISTORY: 11-07-07 M Hewitson
0076 % Creation.
0077 %
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079
0080 function plo = getDefaultPlist()
0081     plo = plist();
0082 end
0083

```

display

PURPOSE ^

DISPLAY implement terminal display for history object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY implement terminal display for history object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY implement terminal display for history object.

CALL:      txt = display(history)

INPUT:     history - history object

OUTPUT:    txt      - cell array with strings to display the history object

M-FILE INFO: Get information about this methods by calling
              >> history.getInfo('display')

              Get information about a specified set-plist by calling:
              >> history.getInfo('display', 'set')

VERSION:   $Id: display.m,v 1.16 2008/09/04 15:29:30 ingo Exp $

HISTORY:   30-01-2007 M Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [history](#) HISTORY History object class constructor.

SUBFUNCTIONS ^

-  [function new_txt = single_cell\(txt_field\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY implement terminal display for history object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY implement terminal display for history object.
0005 %
0006 % CALL:      txt = display(history)
0007 %
0008 % INPUT:     history - history object
0009 %
0010 % OUTPUT:    txt      - cell array with strings to display the history object
0011 %
```

```

0012 % M-FILE INFO: Get information about this methods by calling
0013 % >> history.getInfo('display')
0014 %
0015 % Get information about a specified set-plist by calling:
0016 % >> history.getInfo('display', 'set')
0017 %
0018 % VERSION: $Id: display.m,v 1.16 2008/09/04 15:29:30 ingo Exp $
0019 %
0020 % HISTORY: 30-01-2007 M Hewitson
0021 % Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027 %%% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 hists = utils.helper.collect_objects(varargin(:), 'history');
0034
0035 txt = utils.helper.objdisp(hists);
0036
0037 if nargout == 0
0038     for ii=1:length(txt)
0039         disp(txt{ii});
0040     end
0041 end
0042
0043 varargout{1} = txt;
0044 end
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 % Local Functions %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049
0050 function new_txt = single_cell(txt_field)
0051
0052 new_txt = {};
0053 for ii=1:length(txt_field)
0054     if iscell(txt_field{ii})
0055         hh = single_cell(txt_field{ii});
0056         new_txt(end+1:end+length(hh)) = hh(1:end);
0057     else
0058         new_txt{end+1} = txt_field{ii};
0059     end
0060 end
0061 end
0062
0063 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064 %
0065 % FUNCTION: getInfo
0066 %
0067 % DESCRIPTION: Get Info Object
0068 %
0069 % HISTORY: 11-07-07 M Hewitson
0070 % Creation.
0071 %
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073
0074 function ii = getInfo(varargin)
0075 if nargin == 1 && strcmpi(varargin{1}, 'None')
0076     sets = {};
0077     pl = [];
0078 else
0079     sets = {'Default'};
0080     pl = getDefaultPlist;
0081 end
0082 % Build info object
0083 ii = minfo(mfilename, 'history', '', utils.const.categories.output, '$Id: display.m,v
1.16 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0084 end
0085
0086 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0087 %
0088 % FUNCTION: getDefaultPlist
0089 %
0090 % DESCRIPTION: Get Default Plist
0091 %
0092 % HISTORY: 11-07-07 M Hewitson
0093 % Creation.
0094 %
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096

```

```
0097 function plo = getDefaultPlist()
0098     plo = plist();
0099 end
0100
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

hist2dot

PURPOSE ^

HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz

SYNOPSIS ^

function varargout = hist2dot(varargin)

DESCRIPTION ^

```
HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: HIST2DOT converts a history object to a 'DOT' file suitable for
processing with graphviz (www.graphviz.org).

CALL:      hist2dot(h, 'foo.dot');

INPUT:      h          - history object
            foo.dot    - file name to view the graphic with the programm
                        graphviz

M-FILE INFO: Get information about this methods by calling
            >> history.getInfo('hist2dot')

            Get information about a specified set-plist by calling:
            >> history.getInfo('hist2dot', 'set')

VERSION:    $Id: hist2dot.m,v 1.6 2008/09/04 15:29:30 ingo Exp $

HISTORY:    23-05-2007 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [getNode](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.

This function is called by:

-  [history](#) HISTORY History object class constructor.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % HIST2DOT converts a history object to a 'DOT' file suitable for processing with
graphviz
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: HIST2DOT converts a history object to a 'DOT' file suitable for
0005 % processing with graphviz (www.graphviz.org).
0006 %
```

```

0007 % CALL:      hist2dot(h, 'foo.dot');
0008 %
0009 % INPUT:      h      - history object
0010 %            foo.dot - file name to view the graphic with the programm
0011 %                  graphviz
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %              >> history.getInfo('hist2dot')
0015 %
0016 %              Get information about a specified set-plist by calling:
0017 %              >> history.getInfo('hist2dot', 'set')
0018 %
0019 % VERSION:    $Id: hist2dot.m,v 1.6 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY:    23-05-2007 M Hewitson
0022 %              Creation
0023 %
0024 %%%%%%%%%%%
0025
0026 function varargout = hist2dot(varargin)
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 %%% Set inputs
0035 h      = varargin{1};
0036 filename = varargin{2};
0037
0038
0039 %%% Write .dot file
0040
0041 fd = fopen(filename, 'w+');
0042
0043 %%% write header
0044 fprintf(fd, 'digraph G \n{\n');
0045
0046 [n,a, nodes] = getNodes(h, [], 0, 1, []);
0047
0048 %%% Write block set
0049 fprintf(fd, 'end [label="END"];\n');
0050 for j=1:length(nodes)
0051     node = nodes(j);
0052     % wrap label
0053     % Do some command substitution
0054     fcn = node.names;
0055     shape = 'rectangle';
0056     fsize = 12;
0057     Osize = 24;
0058     opextras = 'style=filled, fillcolor=gray90 fixedsize=true width=0.5';
0059     extras = '';
0060     switch fcn
0061     case 'times'
0062         fcn = '*';
0063         shape = 'circle';
0064         fsize = Osize;
0065         extras = opextras;
0066     case 'mtimes'
0067         fcn = '.*';
0068         shape = 'circle';
0069         fsize = Osize;
0070         extras = opextras;
0071     case 'plus'
0072         fcn = '+';
0073         shape = 'circle';
0074         fsize = Osize;
0075         extras = opextras;
0076     case 'minus'
0077         fcn = '-';
0078         shape = 'circle';
0079         fsize = Osize;
0080         extras = opextras;
0081     case 'rdivide'
0082         fcn = './';
0083         shape = 'circle';
0084         fsize = Osize;
0085         extras = opextras;
0086     case 'mrdivide'
0087         fcn = '/';
0088         shape = 'circle';
0089         fsize = Osize;
0090         extras = opextras;
0091     end
0092

```



```
0093     wstr = utils.prog.wrapstring([fcn char(node.params)], 10);
0094     ss = '';
0095     for s=wstr
0096         ss = [ss '\n' char(s)];
0097     end
0098     ss = ss(3:end);
0099     fprintf(fd, '%s_%d [%s fontsize=%d shape=%s label="%s"];\n', node.names, j, extras,
fsize, shape, ss);
0100 end
0101
0102 %%% Write node list
0103 fprintf(fd, '\n');
0104 fprintf(fd, '\n');
0105
0106 for j=length(nodes):-1:1
0107     % disp('-----')
0108     node = nodes(j);
0109     if node.pn > 0
0110         dst = nodes(node.pn);
0111         fprintf(fd, '%s_%d -> %s_%d;\n', node.names, j, dst.names, dst.n);
0112     else
0113         fprintf(fd, '%s_%d -> end;\n', node.names, j);
0114     end
0115     % disp('-----')
0116 end
0117
0118 %%% close
0119 fprintf(fd, '\n');
0120
0121 %%% Close
0122 fclose(fd);
0123
0124 end
0125
0126
0127
0128 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0129 %                                     Local Functions                                     %
0130 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0131
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133 %
0134 % FUNCTION:      getInfo
0135 %
0136 % DESCRIPTION:  Get Info Object
0137 %
0138 % HISTORY:      11-07-07 M Hewitson
0139 %               Creation.
0140 %
0141 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0142
0143 function ii = getInfo(varargin)
0144     if nargin == 1 && strcmpi(varargin{1}, 'None')
0145         sets = {};
0146         pl = [];
0147     else
0148         sets = {'Default'};
0149         pl = getDefaultPlist;
0150     end
0151     % Build info object
0152     ii = minfo(mfilename, 'history', '', utils.const.categories.output, '$Id: hist2dot.m,v
1.6 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0153 end
0154
0155 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0156 %
0157 % FUNCTION:      getDefaultPlist
0158 %
0159 % DESCRIPTION:  Get Default Plist
0160 %
0161 % HISTORY:      11-07-07 M Hewitson
0162 %               Creation.
0163 %
0164 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0165
0166 function plo = getDefaultPlist()
0167     plo = plist();
0168 end
0169
```

hist2m

PURPOSE ^

HIST2M writes a new m-file that reproduces the analysis described in the history object.

SYNOPSIS ^

```
function cmds = hist2m(varargin)
```

DESCRIPTION ^

```
HIST2M writes a new m-file that reproduces the analysis described in the history object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: HIST2M writes a new m-file that reproduces the analysis described
              in the history object.

CALL:        cmds = hist2m(h);

INPUT:       h      - history object

OUTPUT:      cmds - cell array with the commands to reproduce the data

M-FILE INFO: Get information about this methods by calling
              >> history.getInfo('hist2m')

              Get information about a specified set-plist by calling:
              >> history.getInfo('hist2m', 'set')

VERSION:     $Id: hist2m.m,v 1.30 2008/09/05 14:17:16 hewitson Exp $

HISTORY:     06-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [getNode](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
-  [string](#) STRING writes a command string that can be used to recreate the input history object.

This function is called by:

-  [history](#) HISTORY History object class constructor.

SUBFUNCTIONS ^

-  [function cmd = writeCmd\(name, pl, aon, ains\)](#)
-  [function ps = writePlist\(pl\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % HIST2M writes a new m-file that reproduces the analysis described in the history
object.
```

```

0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: HIST2M writes a new m-file that reproduces the analysis described
0005 %               in the history object.
0006 %
0007 % CALL:        cmds = hist2m(h);
0008 %
0009 % INPUT:       h      - history object
0010 %
0011 % OUTPUT:      cmds - cell array with the commands to reproduce the data
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> history.getInfo('hist2m')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> history.getInfo('hist2m', 'set')
0018 %
0019 % VERSION:     $Id: hist2m.m,v 1.30 2008/09/05 14:17:16 hewitson Exp $
0020 %
0021 % HISTORY:     06-02-2007 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 function cmds = hist2m(varargin)
0027 %
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     cmds = getInfo(varargin{3});
0031     return
0032 end
0033 %
0034 import utils.const.*
0035 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0036 %
0037 % Created contains the epochtime in milliseconds and CREATED2INT define
0038 % the number of the last digits
0039 CREATED2INT = 1e7;
0040 %
0041 % get a node list
0042 utils.helper.msg(msg.PROC1, 'extracting node list from history');
0043 [n,a, nodes] = getNodes(varargin{1}, [], 0, 1, []);
0044 utils.helper.msg(msg.PROC1, 'converting history nodes to commands');
0045 % loop over nodes and convert to commands
0046 i = 1;
0047 cmds = {};
0048 while i <= length(nodes)
0049     v = nodes(i).n;
0050     idx = find([nodes(:).pn] == i);
0051     pl = nodes(i).pl;
0052     aoName = mod(nodes(i).h.proctime, CREATED2INT);
0053     hi = [nodes(idx).h];
0054     iNames = zeros(size(hi));
0055     for j=1:length(hi)
0056         created = hi(j).proctime; %get(hi(j), 'created');
0057         iNames(j) = mod(created, CREATED2INT);
0058     end
0059     cmd = writeCmd(char(nodes(i).names), pl, aoName, iNames);
0060     utils.helper.msg(msg.PROC2, 'wrote command for node %d [%s]', i,
char(nodes(i).names));
0061     cmds = [cmds cellstr(cmd)];
0062     i = i + 1;
0063 end
0064 %
0065 % now find commands that are duplicated after the '=' and remap those
0066 utils.helper.msg(msg.PROC1, 'fixing duplicate commands');
0067 ncmds = length(cmds);
0068 for j = 1:ncmds
0069     cmdj = cmds{j};
0070     % now inspect all other commands prior to this one
0071     for k = j+1:ncmds
0072         cmdk = cmds{k};
0073         if strcmp(cmdj, cmdk)
0074             cmds{j} = '';
0075         end
0076     end
0077 end
0078 %
0079 utils.helper.msg(msg.PROC1, 'writing output line');
0080 % add the final command to produce a_out
0081 alast = deblank(strtok(cmds{1}, '='));
0082 cmds = [cellstr(sprintf('a_out = %s;', alast)) cmds];
0083 end
0084 %
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086 % Local Functions %

```

```

0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091 % FUNCTION:      writeCmd                                     %
0092 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0093 % DESCRIPTION: write a command-line                          %
0094 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096 function cmd = writeCmd(name, pl, aon, ains)
0097
0098     ainsStr = '';
0099     % for i=length(ains):-1:1
0100     ni = length(ains);
0101     for i=1:ni
0102         ainsStr = [ainsStr sprintf('%d', ains(i))];
0103     end
0104
0105     name = strrep(name, '\_', '_');
0106
0107     % look at the input parameters
0108     if isa(pl, 'plist')
0109         ps = writePlist(pl);
0110     else
0111         np = 0;
0112         ps = '';
0113     end
0114     if strcmp(ps, 'plist([])')
0115         ps = '';
0116     end
0117     pstr = deblank(sprintf('%s', ainsStr, ps));
0118
0119     if ~isempty(pstr)
0120         if pstr(end) == ','
0121             pstr = pstr(1:end-1);
0122         end
0123     end
0124     cmd = sprintf('%d = %s(%s);', aon, name, pstr);
0125 end
0126 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0128 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0129 % FUNCTION:      writePlist                                    %
0130 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0131 % DESCRIPTION: write a plist                                  %
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0134 function ps = writePlist(pl)
0135
0136     ps = 'plist(';
0137     np = length(pl.params(:));
0138     for i=1:np
0139         p = pl.params(i);
0140         % if ~isempty(p.val)
0141
0142         %%% char %%%
0143         if ischar(p.val)
0144             ps = [ps sprintf(''%s'', '%s'',', p.key, p.val)];
0145
0146         %%% numeric %%%
0147         elseif isnumeric(p.val)
0148             ps = [ps sprintf(''%s'', [, p.key)   utils.helper.mat2str(p.val) '],'];
0149
0150         %%% specwin %%%
0151         elseif isa(p.val, 'specwin')
0152             w = p.val;
0153             wstr = string(w);
0154             ps = [ps sprintf(''%s'', %s,', p.key, wstr) ];
0155
0156         %%% pzmodel %%%
0157         elseif isa(p.val, 'pzmodel')
0158             pzm = p.val;
0159             fstr = string(pzm);
0160             ps = [ps sprintf(''%s'', %s,', p.key, fstr) ];
0161
0162         %%% time %%%
0163         elseif isa(p.val, 'time')
0164             tt = p.val;
0165             fstr = string(tt);
0166             ps = [ps sprintf(''%s'', %s,', p.key, fstr) ];
0167
0168         %%% provenance %%%
0169         elseif isa(p.val, 'provenance')
0170
0171         %%% history %%%
0172         elseif isa(p.val, 'history')

```

```

0173
0174     %%% plist %%%
0175     elseif isa(p.val, 'plist')
0176         fstr = string(p.val);
0177         ps = [ps sprintf('%s', fstr) ];
0178
0179     %%% pz-object %%%
0180     elseif isa(p.val, 'pz')
0181         fstr = string(p.val);
0182         ps = [ps sprintf(''%s'', %s,', p.key, fstr)];
0183
0184     %%% symbolic math object %%%
0185     elseif isa(p.val, 'sym')
0186         fstr = char(p.val);
0187         ps = [ps sprintf(''%s'', sym(''%s''),', p.key, fstr)];
0188
0189     %%% ltpda_uo %%%
0190     elseif isa(p.val, 'ltpda_uo')
0191         f = p.val;
0192         fstr = string(f);
0193         ps = [ps sprintf(''%s'', %s,', p.key, fstr) ];
0194     elseif isa(p.val, 'unit')
0195         fstr = string(p.val);
0196         ps = [ps sprintf(''%s'', %s,', p.key, fstr)];
0197     elseif iscell(p.val)
0198         fstr = '{}';
0199         for kk = 1: numel(p.val)
0200             if ischar(p.val{kk})
0201                 fstr = sprintf('%s''%s'', ', fstr, p.val{kk});
0202             elseif isnumeric(p.val{kk})
0203                 fstr = sprintf('%s[%s]', fstr, utils.helper.mat2str(p.val{kk}));
0204             elseif isa(p.val{kk}, 'ltpda_obj')
0205                 fstr = sprintf('%s%s, ', fstr, string(p.val{kk}));
0206             else
0207                 error(['### unknown parameter type: ' p.key ' ' class(p.val{kk})]);
0208             end
0209         end
0210         if length(fstr) >= 3, fstr = fstr(1:end-2); end
0211         fstr = [fstr ''];
0212         ps = [ps sprintf(''%s'', %s,', p.key, fstr) ];
0213
0214     else
0215         error(['### unknown parameter type: ' p.key ' ' class(p.val)]);
0216     end
0217 % end
0218 end
0219 if ps(end) == ',',
0220     ps = ps(1:end-1);
0221 end
0222 ps = [ps ')'];
0223 end
0224
0225 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0226 %
0227 % FUNCTION:      getInfo
0228 %
0229 % DESCRIPTION:  Get Info Object
0230 %
0231 % HISTORY:      11-07-07 M Hewitson
0232 %               Creation.
0233 %
0234 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0235
0236 function ii = getInfo(varargin)
0237     if nargin == 1 && strcmpi(varargin{1}, 'None')
0238         sets = {};
0239         pl = [];
0240     else
0241         sets = {'Default'};
0242         pl = getDefaultPlist;
0243     end
0244     % Build info object
0245     ii = minfo(mfilename, 'history', '', utils.const.categories.output, '$Id: hist2m.m,v
1.30 2008/09/05 14:17:16 hewitson Exp $', sets, pl);
0246 end
0247
0248 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0249 %
0250 % FUNCTION:      getDefaultPlist
0251 %
0252 % DESCRIPTION:  Get Default Plist
0253 %
0254 % HISTORY:      11-07-07 M Hewitson
0255 %               Creation.
0256 %
0257 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
0258  
0259 function plo = getDefaultPlist()  
0260     plo = plist();  
0261 end  
0262
```

plot

PURPOSE ^

PLOT plots a history object as a tree diagram.

SYNOPSIS ^

function varargout = plot(varargin)

DESCRIPTION ^

```
PLOT plots a history object as a tree diagram.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PLOT plots a history object as a tree diagram.

NOTE:       Some of the code below is taken from Mathworks's treeplot.m

CALL:       plot (history)
            plot (history, arg)
            plot (axes_handle, ...)

M-FILE INFO: Get information about this methods by calling
            >> history.getInfo('plot')

            Get information about a specified set-plist by calling:
            >> history.getInfo('plot', 'set')

VERSION:    $Id: plot.m,v 1.27 2008/09/05 14:17:01 hewitson Exp $

HISTORY:    02-02-07 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [getNodes](#) GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
-  [getNodes_plot](#) GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
-  [plot](#) PLOT plots a history object as a tree diagram.

This function is called by:

-  [history](#) HISTORY History object class constructor.
-  [plot](#) PLOT plots a history object as a tree diagram.

SUBFUNCTIONS ^

-  function fcname = getFcnName(name)
-  function str = getNodeInputs(invars)
-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```

0001 % PLOT plots a history object as a tree diagram.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PLOT plots a history object as a tree diagram.
0005 %
0006 % NOTE:      Some of the code below is taken from Mathworks's treeplot.m
0007 %
0008 % CALL:      plot (history)
0009 %            plot (history, arg)
0010 %            plot (axes_handle, ...)
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> history.getInfo('plot')
0014 %
0015 %            Get information about a specified set-plist by calling:
0016 %              >> history.getInfo('plot', 'set')
0017 %
0018 % VERSION:   $Id: plot.m,v 1.27 2008/09/05 14:17:01 hewitson Exp $
0019 %
0020 % HISTORY:   02-02-07 M Hewitson
0021 %            Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024 %
0025 function varargout = plot(varargin)
0026
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 import utils.const.*
0035 utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0036
0037 %%% Used call: plot (history, arg)
0038 if isa(varargin{1}, 'history')
0039
0040     figure;
0041     hists = varargin{1};
0042     curr_axes = axes;
0043     if nargin >= 2
0044         varargin = varargin(2:end);
0045     else
0046         varargin = {};
0047     end
0048
0049     %%% Used call: plot (curr_axes_handle, history, arg)
0050 elseif ishandle(varargin{1})
0051
0052     curr_axes = varargin{1};
0053     axes(curr_axes);
0054     hists = varargin{2};
0055     if nargin >= 3
0056         varargin = varargin(3:end);
0057     else
0058         varargin = {};
0059     end
0060
0061 else
0062     error(['### the first input should be a history object or an axes handle' ...
0063         sprintf(' varargin{1} = [%s]', varargin{1})]);
0064 end
0065
0066 %%% collect the parameter and put them into the parameter list
0067 plot_args = {};
0068 pl        = plist();
0069 args      = varargin;
0070
0071 while ~isempty(args)
0072
0073     if isa(args{1}, 'plist')
0074         arg = args{1};
0075         pl = append(pl, arg);
0076         args = args(2:end);
0077
0078         % Add the key-values to the parameter list
0079     elseif length(args) >= 2
0080         if ischar(args{1})
0081             arg = args{1};
0082             val = args{2};
0083             pl = append(pl, param(arg, val));
0084
0085             plot args{end+1} = arg;

```



```

0086         plot_args{end+1} = val;
0087         args = args(3:end);
0088
0089     else
0090         error('### the key [%s] is not from the type ''char'', char(args{1}));
0091     end
0092 else
0093     help(mfilename('fullpath'))
0094     error('### There is no key/value pair left.');
```

```

0095 end
0096
0097 end
0098
0099 %% Comes the call from the browser or from the command window
0100 max_depth = find(pl, 'max_depth');
```

```

0101
0102 %% Output handles for figures and axes
0103 hfig = [];
0104 ax   = [];
0105 at   = [];
0106
0107 nhists = length(hists);
0108 for i=1:nhists
0109
0110     hist = hists(i);
0111     % disp(sprintf('--- plotting history %s', hist.name));
0112
0113     % convert history object to a node list
0114     if ~isempty(max_depth)
0115         [n,a, nodes] = getNodes_plot(hist, [], 0, 1, [], 0, max_depth);
0116     else
0117         [n,a, nodes] = getNodes(hist, [], 0, 1, []);
0118     end
0119     p = [nodes(:).pn];
0120     [x,y,h]=treelayout(p);
0121     f = find(p~=0);
0122     pp = p(f);
0123     X = [x(f); x(pp); repmat(NaN,size(f))];
0124     Y = [y(f); y(pp); repmat(NaN,size(f))];
0125     X = X(:);
0126     Y = Y(:);
0127
0128     % figure
0129     hfig = [hfig gcf];
0130
0131     % axes objects
0132     a = plot(curr_axes, X, Y, 'r--', x, y, 'ro');
0133     % plot(X,Y) --> die linien
0134     if length(a)>1
0135         set(a(2), 'MarkerFaceColor', 'r');
0136         set(a(2), 'MarkerSize', 8);
0137     end
0138     args = plot_args;
0139     while ~isempty(args)
0140         prop = args{1};
0141         val = args{2};
0142         args = args(3:end);
0143         for ii = 1:length(a)
0144             set(a(ii), prop, val);
0145         end
0146     end
0147     ax = [ax a];
0148
0149     % text objects
0150     a = [];
0151     for j=1:length(x)
0152         % node description
0153         fcname = getFcnName(nodes(j).names);
0154         str = ['{\bf\color{blue}' num2str(nodes(j).n) ':{\bf\color{red}' fcname '}' '
char(nodes(j).params)];
0155         nlstr = getNodeInputs(nodes(j).invars);
0156
0157         na = text(x(j), y(j), ...
0158             [utils.prog.wrapstring(strrep(str, '_', '\_'),getappdata(0, 'wrapstringat')) ...
0159             cellstr(['{\color{magenta}' strrep(nlstr, '_', '\_') '}]));
0160         set(na, 'HorizontalAlignment', 'center');
0161         set(na, 'EdgeColor', 'k', 'FontSize', 10);
0162         set(na, 'BackgroundColor', 'w');
0163         set(na, 'Margin', 5);
0164         a = [a na];
0165     end
0166     at = [at a];
0167
0168     % xlabel(['height = ' int2str(h)]);
0169     axis([0 1 0 1]);
0170     box off;

```

```

0171     axis off;
0172
0173     end
0174
0175     % Make outputs
0176     if nargout > 0
0177         varargout{1} = hfig;
0178     end
0179     if nargout > 1
0180         varargout{2} = ax;
0181     end
0182     if nargout > 2
0183         varargout{3} = at;
0184     end
0185 end
0186
0187 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0188 %                                     Local Functions                                     %
0189 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0190
0191 %-----
0192 % compute strings to display for function name
0193 function fcnnname = getFcnName(name)
0194
0195 %     switch name
0196 %     case 'mtimes'
0197 %         fcnnname = 'x';
0198 %     case 'times'
0199 %         fcnnname = 'x';
0200 %     case 'plus'
0201 %         fcnnname = '+';
0202 %     case 'minus'
0203 %         fcnnname = '-';
0204 %     case 'sqrt'
0205 %         fcnnname = '\surd';
0206 %     case 'mrdivide'
0207 %         fcnnname = '/';
0208 %     case 'rdivide'
0209 %         fcnnname = '/';
0210 %     otherwise
0211 %         fcnnname = char(name);
0212 %     end
0213
0214 fcnnname = char(name);
0215 end
0216
0217 %-----
0218 % compute strings to display for inputs to nodes
0219 function str = getNodeInputs(invars)
0220
0221 ni = length(invars);
0222 if ni > 0
0223     str = char(invars{1});
0224     for iv=2:ni
0225         s = char(invars{iv});
0226         if ~strcmp(s, 'pl')
0227             str = [str, 'pl', s];
0228         end
0229     end
0230 else
0231     str = '';
0232 end
0233 end
0234
0235 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0236 %
0237 % FUNCTION:      getInfo
0238 %
0239 % DESCRIPTION:   Get Info Object
0240 %
0241 % HISTORY:      11-07-07 M Hewitson
0242 %               Creation.
0243 %
0244 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0245
0246 function ii = getInfo(varargin)
0247     if nargin == 1 && strcmpi(varargin{1}, 'None')
0248         sets = {};
0249         pl = [];
0250     else
0251         sets = {'Default'};
0252         pl = getDefaultPlist;
0253     end
0254     % Build info object
0255     ii = minfo(mfilename, 'history', '', utils.const.categories.output, '$Id: plot.m,v
1.27 2008/09/05 14:17:01 hewitson Exp $', sets, pl);

```

```
0256 end
0257
0258 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0259 %
0260 % FUNCTION:      getDefaultPlist
0261 %
0262 % DESCRIPTION:  Get Default Plist
0263 %
0264 % HISTORY:      11-07-07 M Hewitson
0265 %                Creation.
0266 %
0267 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0268
0269 function plo = getDefaultPlist()
0270     plo = plist();
0271 end
0272
```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input history object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input history object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate the
             input history object.

CALL:       cmd = string(history_obj)

INPUT:      history_obj - history object

OUTPUT:     cmd          - command string to create the input object

M-FILE INFO: Get information about this methods by calling
              >> history.getInfo('string')

              Get information about a specified set-plist by calling:
              >> history.getInfo('string', 'set')

VERSION:    $Id: string.m,v 1.8 2008/09/04 15:29:30 ingo Exp $

HISTORY:    29-03-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input history object.

This function is called by:

-  [hist2m](#) HIST2M writes a new m-file that reproduces the analysis described in the history object.
-  [history](#) HISTORY History object class constructor.
-  [string](#) STRING writes a command string that can be used to recreate the input history object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input history object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input history object.
```

```

0006 %
0007 % CALL:      cmd = string(history_obj)
0008 %
0009 % INPUT:      history_obj - history object
0010 %
0011 % OUTPUT:     cmd          - command string to create the input object
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> history.getInfo('string')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> history.getInfo('string', 'set')
0018 %
0019 % VERSION:    $Id: string.m,v 1.8 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY:     29-03-2007 M Hewitson
0022 %               Creation
0023 %
0024 %%%%%%%%%%
0025 %
0026 function varargout = string(varargin)
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 objs = utils.helper.collect_objects(varargin(:), 'history');
0035
0036 cmd = '';
0037
0038 for ii = 1:numel(objs)
0039     hh = objs(ii);
0040     m_info = hh.methodInfo;
0041     pl_used = hh.plistUsed;
0042     in_hist = hh.inhists;
0043
0044     %%% Create minfo string
0045     if isempty(m_info)
0046         minfo_str = '[]';
0047     else
0048         minfo_str = string(m_info);
0049     end
0050
0051     %%% Create plist string
0052     if isempty(pl_used)
0053         pl_str = '[]';
0054     else
0055         pl_str = string(pl_used);
0056     end
0057
0058     %%% Create history string
0059     if isempty(in_hist)
0060         hi_str = '[]';
0061     else
0062         hi_str = string(in_hist);
0063     end
0064     if isempty(hi_str)
0065         hi_str = '[]';
0066     end
0067
0068     cmd = [cmd 'history(' minfo_str, ', ' pl_str, ', ' hi_str, ') '];
0069 end
0070
0071 if numel(objs) > 1
0072     cmd = ['[' cmd '']'];
0073 end
0074
0075 varargout{1} = cmd;
0076 end
0077
0078 %%%%%%%%%%
0079 %               Local Functions
0080 %%%%%%%%%%
0081 %
0082 %%%%%%%%%%
0083 %
0084 % FUNCTION:     getInfo
0085 %
0086 % DESCRIPTION:  Get Info Object
0087 %
0088 % HISTORY:      11-07-07 M Hewitson
0089 %               Creation.
0090 %
0091 %%%%%%%%%%

```

```
0092
0093 function ii = getInfo(varargin)
0094     if nargin == 1 && strcmpi(varargin{1}, 'None')
0095         sets = {};
0096         pl = [];
0097     else
0098         sets = {'Default'};
0099         pl = getDefaultPlist;
0100     end
0101     % Build info object
0102     ii = minfo(mfilename, 'history', '', utils.const.categories.output, '$Id: string.m,v
1.8 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0103 end
0104
0105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106 %
0107 % FUNCTION:      getDefaultPlist
0108 %
0109 % DESCRIPTION:  Get Default Plist
0110 %
0111 % HISTORY:      11-07-07 M Hewitson
0112 %                Creation.
0113 %
0114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0115
0116 function plo = getDefaultPlist()
0117     plo = plist();
0118 end
0119
```

provenance

PURPOSE ^

PROVENANCE constructors for provenance class.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
PROVENANCE constructors for provenance class.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION:  PROVENANCE constructors for provenance class.

SUPER CLASSES: ltpda_nuo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
        version          - cvs-version string.

    Protected Properties (read only)
        creator          - current user of the LTPDA toolbox
        ip               - ip address of the creator
        hostname         - hostname of the creator
        os               - used system of the creator
        matlab_version   - MATLAB version
        sigproc_version  - Signal Processing Toolbox version
        symbolic_math_version - Symbolic Math Toolbox version
        ltpda_version    - LTPDA Toolbox version
        version          - version of the constructor

PROVENANCE METHODS:

    Defined Abstract methods:
        char             - returns one character string which represents the object
        copy             - copies an object
        display          - displays an object
        update_struct    - updates a object structure to the current tbx-version

    Public methods:
        string           - writes a command string that can be used to recreate the object

CONSTRUCTORS:

    p = provenance();           - creates an empty provenance object
    p = provenance('creator'); - creates aprovenance object with defiened user

M-FILE INFO:  The following call returns an minfo object that contains
               information about the A0 constructor:
               >> info = provenance.getInfo
               or  >> info = provenance.getInfo('provenance')

               You can get information about class methods by calling:
               >> info = provenance.getInfo(method)
               e.g. >> info = provenance.getInfo('eq')

               You can also restrict the sets of parameters contained in
               the minfo object by calling:
               >> info = provenance.getInfo(method, set)
               e.g. >> info = provenance.getInfo('provenance', 'Set')

VERSION:      $Id: provenance.m,v 1.40 2008/09/03 16:35:40 hewitson Exp $

HISTORY:      07-05-2007 M Hewitson
               Creation

SEE ALSO:     ltpda_obj, ltpda_nuo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  **char** CHAR convert a provenance object into a string.
-  **copy** COPY Make copy of provenance objects depending of the second input
-  **provenance** PROVENANCE constructors for provenance class.
-  **update_struct** UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  **copy** COPY Make copy of provenance objects depending of the second input
-  **provenance** PROVENANCE constructors for provenance class.

SUBFUNCTIONS [^]

-  **function** obj = provenance(varargin)
-  **function** out = VEROUT()
-  **function** ii = getInfo(varargin)
-  **function** out = SETS()
-  **function** out = getDefaultPlist(set)

SOURCE CODE [^]

```

0001 % PROVENANCE constructors for provenance class.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION:  PROVENANCE constructors for provenance class.
0005 %
0006 % SUPER CLASSES: ltpda_nuo < ltpda_obj
0007 %
0008 % PROPERTIES:
0009 %
0010 %     Inherit Properties (read only)
0011 %         version          - cvs-version string.
0012 %
0013 %     Protected Properties (read only)
0014 %         creator           - current user of the LTPDA toolbox
0015 %         ip                - ip address of the creator
0016 %         hostname         - hostname of the creator
0017 %         os                - used system of the creator
0018 %         matlab_version   - MATLAB version
0019 %         sigproc_version  - Signal Processing Toolbox version
0020 %         symbolic_math_version - Symbolic Math Toolbox version
0021 %         ltpda_version    - LTPDA Toolbox version
0022 %         version          - version of the constructor
0023 %
0024 % PROVENANCE METHODS:
0025 %
0026 %     Defined Abstract methods:
0027 %         char              - returns one character string which represents the object
0028 %         copy              - copies an object
0029 %         display           - displays an object
0030 %         update_struct     - updates a object structure to the current tbx-version
0031 %
0032 %     Public methods:
0033 %         string            - writes a command string that can be used to recreate the object
0034 %
0035 % CONSTRUCTORS:
0036 %
0037 %         p = provenance();          - creates an empty provenance object
0038 %         p = provenance('creator'); - creates aprovenance object with defiened user
0039 %
0040 % M-FILE INFO:  The following call returns an minfo object that contains
0041 %                information about the A0 constructor:
0042 %                >> info = provenance.getInfo
0043 %                or   >> info = provenance.getInfo('provenance')
0044 %
0045 %                You can get information about class methods by calling:
0046 %                >> info = provenance.getInfo(method)
0047 %                e.g. >> info = provenance.getInfo('eq')

```



```

0048 %
0049 %
0050 % You can also restrict the sets of parameters contained in
0051 % the minfo object by calling:
0052 %     >> info = provenance.getInfo(method, set)
0053 % e.g. >> info = provenance.getInfo('provenance', 'Set')
0054 %
0055 % VERSION: $Id: provenance.m,v 1.40 2008/09/03 16:35:40 hewitson Exp $
0056 %
0057 % HISTORY: 07-05-2007 M Hewitson
0058 %           Creation
0059 %
0060 % SEE ALSO: ltpda_obj, ltpda_nuo
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062 %
0063 classdef provenance < ltpda_nuo
0064 %
0065 %----- Private read-only Properties -----
0066 %----- Private read-only Properties -----
0067 %
0068 properties (SetAccess = private)
0069     creator =
0070 char(java.lang.System.getProperties.getProperty('user.name'));
0071     ip = char(getHostAddress(java.net.InetAddress.getLocalHost));
0072     hostname = char(getHostName(java.net.InetAddress.getLocalHost));
0073     os = computer;
0074     matlab_version = getappdata(0, 'matlab_version');
0075     sigproc_version = getappdata(0, 'sigproc_version');
0076     symbolic_math_version = getappdata(0, 'ltpda_symbolic_math');
0077     ltpda_version = getappdata(0, 'ltpda_version');
0078 end
0079 properties (SetAccess = protected)
0080     version = '$Id: provenance.m,v 1.40 2008/09/03 16:35:40 hewitson Exp $';
0081 end
0082 %
0083 %----- Declaration of public methods -----
0084 %----- Declaration of public methods -----
0085 %
0086 methods
0087 %
0088 %----- Property rules -----
0089 %----- Property rules -----
0090 %
0091 %
0092 %----- Constructor -----
0093 % Constructor
0094 %-----
0095 function obj = provenance(varargin)
0096 %
0097 import utils.const.*
0098 utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);
0099 %
0100 %% Call superclass
0101 obj = obj@ltpda_nuo(varargin{:});
0102 %
0103 %%%%%%%%% Set default values %%%%%%%%%
0104 %
0105 if nargin == 1
0106     if isstruct(varargin{1})
0107         utils.helper.msg(msg.OPROC1, 'constructing from struct');
0108         %%%%%%%%% prov = provenance(structure) %%%%%%%%%
0109         %%%%%%%%% necessary for readxml %%%%%%%%%
0110 %
0111 %% Set properties which are declared in this class
0112 prov_struc = varargin{1};
0113 %
0114 obj.creator = prov_struc.creator;
0115 obj.ip = prov_struc.ip;
0116 obj.hostname = prov_struc.hostname;
0117 obj.os = prov_struc.os;
0118 obj.matlab_version = prov_struc.matlab_version;
0119 obj.sigproc_version = prov_struc.sigproc_version;
0120 obj.symbolic_math_version = prov_struc.symbolic_math_version;
0121 obj.ltpda_version = prov_struc.ltpda_version;
0122 obj.version = prov_struc.version;
0123 %
0124 elseif ischar(varargin{1})
0125     obj.creator = varargin{1};
0126 elseif isa(varargin{1}, 'provenance')
0127     utils.helper.msg(msg.OPROC1, 'copy constructor');
0128     obj = copy(varargin{1}, 1);
0129 else
0130     error('### Unknown constructor');
0131 end
0132 end

```

```
0133
0134     end % End of constructor
0135
0136 end % End public methods
0137
0138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0139 %                                     Methods (public) %
0140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141
0142 methods (Access = public)
0143     varargout = copy(varargin)
0144 end
0145
0146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0147 %                                     Methods (protected) %
0148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0149
0150 methods (Access = protected)
0151 end
0152
0153 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0154 %                                     Methods (Static, Public) %
0155 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0156
0157 methods (Static = true)
0158
0159     varargout = update_struct(varargin);
0160
0161     function out = VEROUT()
0162         out = '$Id: provenance.m,v 1.40 2008/09/03 16:35:40 hewitson Exp $';
0163     end
0164
0165     function ii = getInfo(varargin)
0166         ii = utils.helper.generic_getInfo(varargin{:}, 'provenance');
0167     end
0168
0169     function out = SETS()
0170         out = {'Default'};
0171     end
0172
0173     function out = getDefaultPlist(set)
0174         switch set
0175             case 'Default'
0176                 out = plist();
0177             otherwise
0178                 out = plist();
0179         end
0180     end
0181
0182 end % End static methods
0183
0184 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0185 %                                     Methods (Static, Private) %
0186 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0187
0188 methods (Static = true, Access = private)
0189 end % End static private methods
0190
0191 end % End classdef
0192
```

copy

PURPOSE ^

COPY Make copy of provenance objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

COPY Make copy of provenance objects depending of the second input
DESCRIPTION: COPY Make copy of provenance objects depending of the second input
CALL: objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
>> provenance.getInfo('copy')
VERSION: \$Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp \$
HISTORY: 11-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 provenance PROVENANCE constructors for provenance class.

This function is called by:

 provenance PROVENANCE constructors for provenance class.

SOURCE CODE ^

```
0001 % COPY Make copy of provenance objects depending of the second input
0002 %
0003 %
0004 % DESCRIPTION: COPY Make copy of provenance objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %             >> provenance.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %             Creation
0015 %
0016 %
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('provenance');
0022         s = size(old);
0023         obj(s(1),s(2)) = provenance;
0024         for kk=1:numel(old)
0025             utils.helper.msg(utils.const.msg.OPROC2, 'copying provenance');
```

```
0026     for jj=1:length(ps)
0027         obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028     end
0029     % reset on copy
0030     obj(kk).version = provenance.VEROUT;
0031 end
0032 else
0033     obj = old;
0034 end
0035 varargout{1} = obj;
0036 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

char

PURPOSE ^

CHAR convert a provenance object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a provenance object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a provenance object into a string.

CALL:      string = char(prov)

M-FILE INFO: Get information about this methods by calling
              >> provenance.getInfo('char')

              Get information about a specified set-plist by calling:
              >> provenance.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.10 2008/09/04 15:29:31 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [provenance](#) PROVENANCE constructors for provenance class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a provenance object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a provenance object into a string.
0005 %
0006 % CALL:      string = char(prov)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> provenance.getInfo('char')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> provenance.getInfo('char', 'set')
0013 %
0014 % VERSION:    $Id: char.m,v 1.10 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:    04-02-2007 M Hewitson
0017 %              Creation
```

```

0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 % Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'provenance');
0030
0031 pstr = '';
0032 for ii = 1:numel(objs)
0033     if ~isempty(pstr)
0034         pstr = [pstr ' | '];
0035     end
0036
0037     pstr = [pstr sprintf('created by %s@%s[%s] on %s/%s/%s', ...
0038         objs(ii).creator, ...
0039         objs(ii).hostname, ...
0040         objs(ii).ip, ...
0041         objs(ii).os, ...
0042         objs(ii).matlab_version, ...
0043         objs(ii).ltpda_version)];
0044 end
0045
0046 %% Prepare output
0047 varargout{1} = pstr;
0048 end
0049
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %                               Local Functions                               %
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %
0056 % FUNCTION:      getInfo
0057 %
0058 % DESCRIPTION:  Get Info Object
0059 %
0060 % HISTORY:      11-07-07 M Hewitson
0061 %               Creation.
0062 %
0063 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064
0065 function ii = getInfo(varargin)
0066     if nargin == 1 && strcmpi(varargin{1}, 'None')
0067         sets = {};
0068         pl = [];
0069     else
0070         sets = {'Default'};
0071         pl = getDefaultPlist;
0072     end
0073     % Build info object
0074     ii = minfo(mfilename, 'provenance', '', utils.const.categories.output, '$Id: char.m,v
1.10 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0075 end
0076
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %
0079 % FUNCTION:      getDefaultPlist
0080 %
0081 % DESCRIPTION:  Get Default Plist
0082 %
0083 % HISTORY:      11-07-07 M Hewitson
0084 %               Creation.
0085 %
0086 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0087
0088 function plo = getDefaultPlist()
0089     plo = plist();
0090 end
0091

```

display

PURPOSE ^

DISPLAY overload terminal display for provenance objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overload terminal display for provenance objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overload terminal display for provenance objects.

M-FILE INFO: Get information about this methods by calling
              >> provenance.getInfo('display')

              Get information about a specified set-plist by calling:
              >> provenance.getInfo('display', 'set')

VERSION:      $Id: display.m,v 1.12 2008/09/04 15:29:31 ingo Exp $

HISTORY: 07-05-2007 M Hewitson
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % DISPLAY overload terminal display for provenance objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overload terminal display for provenance objects.
0005 %
0006 % M-FILE INFO: Get information about this methods by calling
0007 %              >> provenance.getInfo('display')
0008 %
0009 %              Get information about a specified set-plist by calling:
0010 %              >> provenance.getInfo('display', 'set')
0011 %
0012 % VERSION:      $Id: display.m,v 1.12 2008/09/04 15:29:31 ingo Exp $
0013 %
0014 % HISTORY: 07-05-2007 M Hewitson
0015 %          Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 function varargout = display(varargin)
0020 %
0021 % Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
```

```
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 % Get minfo objects
0028 objs = utils.helper.collect_objects(varargin(:), 'provenance');
0029
0030 % get display text
0031 txt = utils.helper.objdisp(objs);
0032
0033 % display the objects
0034 if nargin > 0
0035     varargout{1} = txt;
0036 elseif nargin == 0;
0037     for j=1:numel(txt)
0038         disp(txt{j});
0039     end
0040 end
0041
0042 end
0043
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                                     Local Functions
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:  Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %               Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'provenance', '', utils.const.categories.output, '$Id:
display.m,v 1.12 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0069 end
0070
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:  Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %               Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081
0082 function plo = getDefaultPlist()
0083     plo = plist();
0084 end
0085
```


string

PURPOSE ^

STRING writes a command string that can be used to recreate the input provenance object.

SYNOPSIS ^

```
function cmd = string(varargin)
```

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input provenance object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate the
             input provenance object.

CALL:       cmd = string(obj)

INPUT:      obj - provenance object

OUTPUT:     cmd - command string to create the input object

M-FILE INFO: Get information about this methods by calling
              >> provenance.getInfo('string')

              Get information about a specified set-plist by calling:
              >> provenance.getInfo('string', 'set')

VERSION:    $Id: string.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:    15-02-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input provenance
0002 object.
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input provenance object.
0006 %
0007 % CALL:       cmd = string(obj)
0008 %
0009 % INPUT:      obj - provenance object
0010 %
0011 % OUTPUT:     cmd - command string to create the input object
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> provenance.getInfo('string')
```

```
0015 %
0016 %           Get information about a specified set-plist by calling:
0017 %           >> provenance.getInfo('string', 'set')
0018 %
0019 % VERSION:      $Id: string.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0020 %
0021 % HISTORY:      15-02-2008 Diepholz
0022 %               Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 function cmd = string(varargin)
0027
0028 % Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     cmd = getInfo(varargin{3});
0031     return
0032 end
0033
0034 objs = utils.helper.collect_objects(varargin(:), 'provenance');
0035
0036 %% Wrap the command only in bracket if the there are more than one object
0037 if length(objs) > 1
0038     cmd = '[';
0039 else
0040     cmd = '';
0041 end
0042
0043 for j=1:length(objs)
0044     creator = objs(j).creator;
0045     creator = strcat('Recreated from: ', creator);
0046     cmd = [cmd 'provenance('' creator '') '];
0047 end
0048
0049 if length(objs) > 1
0050     cmd = [cmd '']];
0051 end
0052 end
0053
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %                               Local Functions                               %
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059 %
0060 % FUNCTION:      getInfo
0061 %
0062 % DESCRIPTION:   Get Info Object
0063 %
0064 % HISTORY:      11-07-07 M Hewitson
0065 %               Creation.
0066 %
0067 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0068
0069 function ii = getInfo(varargin)
0070     if nargin == 1 && strcmpi(varargin{1}, 'None')
0071         sets = {};
0072         pl = [];
0073     else
0074         sets = {'Default'};
0075         pl = getDefaultPlist;
0076     end
0077     % Build info object
0078     ii = minfo(mfilename, 'provenance', '', utils.const.categories.output, '$Id:
string.m,v 1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0079 end
0080
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %
0083 % FUNCTION:      getDefaultPlist
0084 %
0085 % DESCRIPTION:   Get Default Plist
0086 %
0087 % HISTORY:      11-07-07 M Hewitson
0088 %               Creation.
0089 %
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091
0092 function plo = getDefaultPlist()
0093     plo = plist();
0094 end
0095
```

[mux](#)

PURPOSE [^]

MUX concatenate params into a vector.

SYNOPSIS [^]

function varargout = mux(varargin)

DESCRIPTION [^]

```
MUX concatenate params into a vector.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MUX concatenate params into a vector.

CALL:      param_vec = mux(p1, p2);

M-FILE INFO: Get information about this methods by calling
              >> param.getInfo('mux')

              Get information about a specified set-plist by calling:
              >> param.getInfo('mux', 'set')

VERSION:    $Id: mux.m,v 1.10 2008/09/04 15:29:31 ingo Exp $

HISTORY:    27-03-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [copy](#) COPY Make copy of param objects depending of the second input

This function is called by:

 [param](#) PARAM Parameter object class constructor.

SUBFUNCTIONS [^]

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE [^]

```
0001 % MUX concatenate params into a vector.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MUX concatenate params into a vector.
0005 %
0006 % CALL:      param_vec = mux(p1, p2);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> param.getInfo('mux')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> param.getInfo('mux', 'set')
0013 %
0014 % VERSION:    $Id: mux.m,v 1.10 2008/09/04 15:29:31 ingo Exp $
```

```
0015 %
0016 % HISTORY:      27-03-2007 M Hewitson
0017 %              Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = mux(varargin)
0022
0023     %% Check if this is a call for parameters
0024     if utils.helper.isinfocall(varargin{:})
0025         varargout{1} = getInfo(varargin{3});
0026         return
0027     end
0028
0029     objs = utils.helper.collect_objects(varargin(:), 'param');
0030
0031     p_list = [];
0032     for ii = 1:numel(objs)
0033         %% decide whether we modify the first plist, or create a new one.
0034         objs(ii) = copy(objs(ii), nargout);
0035
0036         p_list = [p_list objs(ii)];
0037     end
0038
0039     varargout{1} = p_list;
0040
0041 end
0042
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044 %                               Local Functions                               %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048 %
0049 % FUNCTION:      getInfo
0050 %
0051 % DESCRIPTION:   Get Info Object
0052 %
0053 % HISTORY:      11-07-07 M Hewitson
0054 %              Creation.
0055 %
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 function ii = getInfo(varargin)
0059     if nargin == 1 && strcmpi(varargin{1}, 'None')
0060         sets = {};
0061         pl   = [];
0062     else
0063         sets = {'Default'};
0064         pl   = getDefaultPlist;
0065     end
0066     % Build info object
0067     ii = minfo(mfilename, 'param', '', utils.const.categories.helper, '$Id: mux.m,v 1.10
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0068 end
0069
0070 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0071 %
0072 % FUNCTION:      getDefaultPlist
0073 %
0074 % DESCRIPTION:   Get Default Plist
0075 %
0076 % HISTORY:      11-07-07 M Hewitson
0077 %              Creation.
0078 %
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080
0081 function plo = getDefaultPlist()
0082     plo = plist();
0083 end
0084
```

copy

PURPOSE ^

COPY Make copy of param objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of param objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of param objects depending of the second input
CALL:      objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
              >> param.getInfo('copy')
VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
HISTORY:    11-07-2008 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [param](#) PARAM Parameter object class constructor.

This function is called by:

-  [mux](#) MUX concatenate params into a vector.
-  [param](#) PARAM Parameter object class constructor.
-  [setKey](#) SETKEY Set the property 'key'.
-  [setKeyVal](#) SETKEYVAL Set the properties 'key' and 'val'
-  [setVal](#) SETVAL Set the property 'val'.

SOURCE CODE ^

```
0001 % COPY Make copy of param objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of param objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> param.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
```

```
0019
0020 if deepcopy
0021     ps = properties('param');
0022     s = size(old);
0023     obj(s(1),s(2)) = param;
0024     for kk=1:numel(old)
0025         utils.helper.msg(utils.const.msg.OPROC2, 'copying param');
0026         for jj=1:length(ps)
0027             obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028         end
0029         % reset on copy
0030         obj(kk).version = param.VEROUT;
0031     end
0032 else
0033     obj = old;
0034 end
0035 varargout{1} = obj;
0036 end
0037
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setKey

PURPOSE ^

SETKEY Set the property 'key'.

SYNOPSIS ^

function varargout = setKey(varargin)

DESCRIPTION ^

SETKEY Set the property 'key'.

DESCRIPTION: Set the property 'key'.

CALL:
obj = obj.setKey('new key');
obj = obj.setKey(plist('key', 'new key'));
obj = setKey(obj, 'new key');

INPUTS:
obj - can be a vector, matrix, list, or a mix of them.
pl - to set the key with a plist specify only one plist with only one key-word 'key'.

M-FILE INFO: Get information about this methods by calling
>> param.getInfo('setKey')

Get information about a specified set-plist by calling:
>> param.getInfo('setKey', 'set')

VERSION: \$Id: setKey.m,v 1.6 2008/09/04 15:29:31 ingo Exp \$

HISTORY: 27-05-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of param objects depending of the second input

This function is called by:

 [param](#) PARAM Parameter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETKEY Set the property 'key'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'key'.
0005 %
0006 % CALL:      obj = obj.setKey('new key');
0007 %           obj = obj.setKey(plist('key', 'new key'));
0008 %           obj = setKey(obj, 'new key');
```

```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %             pl  - to set the key with a plist specify only one plist with
0012 %                   only one key-word 'key'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> param.getInfo('setKey')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> param.getInfo('setKey', 'set')
0019 %
0020 % VERSION:     $Id: setKey.m,v 1.6 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %               Creation
0024 %
0025 %%%%%%%%%%%%%%
0026
0027 function varargout = setKey(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037
0038     %%% decide whether we modify the first plist, or create a new one.
0039     varargin{1} = copy(varargin{1}, nargout);
0040
0041     varargin{1}.key = varargin{2};
0042     varargout{1} = varargin{1};
0043     return
0044 end
0045
0046 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'param');
0047 [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0048
0049 vals = rest;
0050
0051 if length(vals) ~=1
0052
0053     if length(pls) == 1
0054         vals = {};
0055         if nparams(pls) == 1 && isparam(pls, 'key')
0056
0057             %%% If the plist contains only one parameter with the key-word 'key'
0058             %%% then set 'key' to the value in the plist
0059             vals{1} = find(pls(1), 'key');
0060
0061         else
0062             vals{1} = pls;
0063         end
0064     elseif length(pls) > 1
0065         error('### To set the 'key' please specify only one plist')
0066     else
0067         error('### Please specify [only one] key.')
0068     end
0069 end
0070
0071 %%% Set the Key
0072 for ii = 1:numel(objs)
0073
0074     %%% decide whether we modify the first plist, or create a new one.
0075     objs(ii) = copy(objs(ii), nargout);
0076
0077     objs(ii).key = vals{1};
0078 end
0079
0080 %%% Prepare output
0081 varargout{1} = objs;
0082
0083 end
0084
0085 %%%%%%%%%%%%%%
0086 %                               Local Functions                               %
0087 %%%%%%%%%%%%%%
0088
0089 %%%%%%%%%%%%%%
0090 %
0091 % FUNCTION:      getInfo
0092 %
0093 % DESCRIPTION:  Get Info Object
0094 %

```



```
0095 % HISTORY:      11-07-07 M Hewitson
0096 %
0097 %
0098 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0099
0100 function ii = getInfo(varargin)
0101     if nargin == 1 && strcmpi(varargin{1}, 'None')
0102         sets = {};
0103         pl    = [];
0104     else
0105         sets = {'Default'};
0106         pl    = getDefaultPlist;
0107     end
0108     % Build info object
0109     ii = minfo(mfilename, 'param', '', utils.const.categories.internal, '$Id: setKey.m,v
1.6 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0110 end
0111
0112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113 %
0114 % FUNCTION:      getDefaultPlist
0115 %
0116 % DESCRIPTION:   Get Default Plist
0117 %
0118 % HISTORY:      11-07-07 M Hewitson
0119 %
0120 %
0121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122
0123 function plo = getDefaultPlist()
0124     plo = plist('key', '');
0125 end
0126
```

setKeyVal

PURPOSE ^

SETKEYVAL Set the properties 'key' and 'val'

SYNOPSIS ^

function varargout = setKeyVal(varargin)

DESCRIPTION ^

```
SETKEYVAL Set the properties 'key' and 'val'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the properties 'key' and 'val'

CALL:
    obj = obj.setKeyVal('new key', 'new val');
    obj = obj.setKeyVal(plist('key', 'new key', 'val', 'new val'));
    obj = setKeyVal(obj, 'new key', 'new val');

M-FILE INFO: Get information about this methods by calling
    >> param.getInfo('setKeyVal')

    Get information about a specified set-plist by calling:
    >> param.getInfo('setKeyVal', 'set')

VERSION:      $Id: setKeyVal.m,v 1.6 2008/09/04 15:29:31 ingo Exp $

HISTORY:      27-05-2008 Diepholz
                Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of param objects depending of the second input

This function is called by:

 [param](#) PARAM Parameter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETKEYVAL Set the properties 'key' and 'val'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the properties 'key' and 'val'
0005 %
0006 % CALL:
0007 %     obj = obj.setKeyVal('new key', 'new val');
0008 %     obj = obj.setKeyVal(plist('key', 'new key', 'val', 'new val'));
0009 %     obj = setKeyVal(obj, 'new key', 'new val');
0010 % M-FILE INFO: Get information about this methods by calling
0011 %     >> param.getInfo('setKeyVal')
0012 %
```

```

0013 %           Get information about a specified set-plist by calling:
0014 %           >> param.getInfo('setKeyVal', 'set')
0015 %
0016 % VERSION:      $Id: setKeyVal.m,v 1.6 2008/09/04 15:29:31 ingo Exp $
0017 %
0018 % HISTORY:      27-05-2008 Diepholz
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 function varargout = setKeyVal(varargin)
0024
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 %%% Internal call: Only one object + don't look for a plist
0032 if strcmp(varargin{end}, 'internal')
0033
0034     %%% decide whether we modify the first plist, or create a new one.
0035     varargin{1} = copy(varargin{1}, nargout);
0036
0037     varargin{1}.key = varargin{2};
0038     varargin{1}.val = varargin{3};
0039     varargout{1} = varargin{1};
0040     return
0041 end
0042
0043 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'param');
0044 [pls, invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0045
0046 vals = rest;
0047
0048 if length(vals) ~= 2
0049
0050     if length(pls) == 1
0051         if nparams(pls(1)) == 2    && isparam(pls(1), 'val') && isparam(pls(1), 'key')
0052
0053             %%% If the plist contains only two parameter with the key-word 'key' and 'val'
0054             %%% then set key/val to these values.
0055             vals = cell(1,2);
0056             vals{1} = find(pls(1), 'key');
0057             vals{2} = find(pls(1), 'val');
0058
0059         else
0060             error('### To set the key/val pair please specify only one plist with the key-
words 'key'' and 'val''')
0061         end
0062     else
0063         error('### Please specify [only one] key/value pair.')
0064     end
0065 end
0066
0067 %%% Set the Val
0068 for ii = 1:numel(objs)
0069
0070     %%% decide whether we modify the first plist, or create a new one.
0071     objs(ii) = copy(objs(ii), nargout);
0072
0073     objs(ii).key = vals{1};
0074     objs(ii).val = vals{2};
0075 end
0076
0077 %%% Prepare output
0078 varargout{1} = objs;
0079
0080 end
0081
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083 %                               Local Functions                               %
0084 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0085
0086 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0087 %
0088 % FUNCTION:      getInfo
0089 %
0090 % DESCRIPTION:  Get Info Object
0091 %
0092 % HISTORY:      11-07-07 M Hewitson
0093 %                Creation.
0094 %
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096
0097 function ii = getInfo(varargin)

```

```
0098     if nargin == 1 && strcmpi(varargin{1}, 'None')
0099         sets = {};
0100         pl   = [];
0101     else
0102         sets = {'Default'};
0103         pl   = getDefaultPlist;
0104     end
0105     % Build info object
0106     ii = minfo(mfilename, 'param', '', utils.const.categories.internal, '$Id:
setKeyVal.m,v 1.6 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0107 end
0108
0109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0110 %
0111 % FUNCTION:      getDefaultPlist
0112 %
0113 % DESCRIPTION:  Get Default Plist
0114 %
0115 % HISTORY:      11-07-07 M Hewitson
0116 %               Creation.
0117 %
0118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0119
0120 function plo = getDefaultPlist()
0121     plo = plist('key', '', 'val', '');
0122 end
0123
```

setVal

PURPOSE ^

SETVAL Set the property 'val'.

SYNOPSIS ^

function varargout = setVal(varargin)

DESCRIPTION ^

SETVAL Set the property 'val'.
%%%

DESCRIPTION: Set the property 'val'.

CALL:
obj = obj.setVal('new val');
obj = obj.setVal(plist('val', 'new val'));
obj = setVal(obj, 'new val');

INPUTS:
obj - can be a vector, matrix, list, or a mix of them.
pl - to set the val with a plist specify only one plist with
only one key-word 'val'.

M-FILE INFO: Get information about this methods by calling
>> param.getInfo('setVal')

Get information about a specified set-plist by calling:
>> param.getInfo('setVal', 'set')

VERSION: \$Id: setVal.m,v 1.6 2008/09/04 15:29:31 ingo Exp \$

HISTORY: 27-05-2008 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of param objects depending of the second input

This function is called by:

 [param](#) PARAM Parameter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETVAL Set the property 'val'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'val'.
0005 %
0006 % CALL:      obj = obj.setVal('new val');
0007 %           obj = obj.setVal(plist('val', 'new val'));
0008 %           obj = setVal(obj, 'new val');
```

```

0009 %
0010 % INPUTS:      obj - can be a vector, matrix, list, or a mix of them.
0011 %            pl  - to set the val with a plist specify only one plist with
0012 %                   only one key-word 'val'.
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 %             >> param.getInfo('setVal')
0016 %
0017 %             Get information about a specified set-plist by calling:
0018 %             >> param.getInfo('setVal', 'set')
0019 %
0020 % VERSION:     $Id: setVal.m,v 1.6 2008/09/04 15:29:31 ingo Exp $
0021 %
0022 % HISTORY:     27-05-2008 Diepholz
0023 %              Creation
0024 %
0025 %%%%%%%%%%%%%%
0026
0027 function varargout = setVal(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 %%% Internal call: Only one object + don't look for a plist
0036 if strcmp(varargin{end}, 'internal')
0037
0038     %%% decide whether we modify the first plist, or create a new one.
0039     varargin{1} = copy(varargin{1}, nargout);
0040
0041     varargin{1}.val = varargin{2};
0042     varargout{1} = varargin{1};
0043     return
0044 end
0045
0046 [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'param');
0047 [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');
0048
0049 vals = rest;
0050
0051 if length(vals) ~=1
0052
0053     if length(pls) == 1
0054         vals = {};
0055         if nparams(pls) == 1 && isparam(pls, 'val')
0056
0057             %%% If the plist contains only one parameter with the key-word 'val'
0058             %%% then set 'val' to the value in the plist
0059             vals{1} = find(pls, 'val');
0060
0061         else
0062             vals{1} = pls;
0063         end
0064     elseif length(pls) > 1
0065         error('### To set the 'val' please specify only one plist')
0066     else
0067         error('### Please specify [only one] value.')
0068     end
0069 end
0070
0071 %%% Set the Val
0072 for ii = 1:numel(objs)
0073
0074     %%% decide whether we modify the first plist, or create a new one.
0075     objs(ii) = copy(objs(ii), nargout);
0076
0077     objs(ii).val = vals{1};
0078 end
0079
0080 %%% Prepare output
0081 varargout{1} = objs;
0082 end
0083
0084 %%%%%%%%%%%%%%
0085 %
0086 % FUNCTION:     getInfo
0087 %
0088 % DESCRIPTION: Get Info Object
0089 %
0090 % HISTORY:     11-07-07 M Hewitson
0091 %              Creation.
0092 %
0093 %%%%%%%%%%%%%%
0094

```

```
0095 function ii = getInfo(varargin)
0096     if nargin == 1 && strcmpi(varargin{1}, 'None')
0097         sets = {};
0098         pl = [];
0099     else
0100         sets = {'Default'};
0101         pl = getDefaultPlist;
0102     end
0103     % Build info object
0104     ii = minfo(mfilename, 'param', '', utils.const.categories.internal, '$Id: setVal.m,v
1.6 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0105 end
0106
0107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108 %
0109 % FUNCTION:      getDefaultPlist
0110 %
0111 % DESCRIPTION:  Get Default Plist
0112 %
0113 % HISTORY:      11-07-07 M Hewitson
0114 %               Creation.
0115 %
0116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0117
0118 function plo = getDefaultPlist()
0119     plo = plist('val', '');
0120 end
0121
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a param object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: CHAR convert a param object into a string.
CALL:       string = char(obj)
M-FILE INFO: Get information about this methods by calling
              >> param.getInfo('char')
              Get information about a specified set-plist by calling:
              >> param.getInfo('char', 'set')
VERSION:    $Id: char.m,v 1.17 2008/09/04 15:29:31 ingo Exp $
HISTORY:    04-02-2007 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a param object into a string.

This function is called by:

-  [char](#) CHAR convert a param object into a string.
-  [display](#) DISPLAY display a parameter
-  [display2](#) DISPLAY display a parameter
-  [param](#) PARAM Parameter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a param object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a param object into a string.
0005 %
0006 % CALL:       string = char(obj)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> param.getInfo('char')
```



```

0010 %
0011 %           Get information about a specified set-plist by calling:
0012 %           >> param.getInfo('char', 'set')
0013 %
0014 % VERSION:      $Id: char.m,v 1.17 2008/09/04 15:29:31 ingo Exp $
0015 %
0016 % HISTORY:      04-02-2007 M Hewitson
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 objs = utils.helper.collect_objects(varargin(:), 'param');
0030
0031 pstr = '';
0032 for ii = 1:numel(objs)
0033
0034     pp = objs(ii);
0035
0036     if isempty(pp.key)
0037         pp.key = ' ';
0038     end
0039
0040     %%%%%%%%% cell %%%%%%%%%
0041     if iscell(pp.val)
0042         pstr = [pstr pp.key '={'];
0043         for ii = 1:length(pp.val)
0044             if ischar(pp.val{ii})
0045                 pstr = [pstr ' ', pp.val{ii} ' ', ' '];
0046             elseif isnumeric(pp.val{ii})
0047                 pstr = [pstr ' ', mat2str(pp.val{ii},2) ' ', ' '];
0048             elseif isa(pp.val{ii}, 'sym')
0049                 pstr = [pstr ' ', char(pp.val{ii}) ' ', ' '];
0050             else
0051                 pstr = [pstr ' ', class(pp.val{ii}) '-object, '];
0052             end
0053         end
0054         if strcmp(pstr(end-1:end), ', ')
0055             pstr = [pstr(1:end-2) '}', ' '];
0056         else
0057             pstr = [pstr '}', ' '];
0058         end
0059
0060     %%%%%%%%% character %%%%%%%%%
0061     elseif ischar(pp.val)
0062         sval = char(pp.val);
0063         if isempty(sval)
0064             sval = [''];
0065         elseif length(sval) >= 10
0066             sval = [sval(1:10) '...'];
0067         end
0068         pstr = [pstr pp.key '=' sval ' ', ' '];
0069
0070     %%%%%%%%% numbers %%%%%%%%%
0071     elseif isnumeric(pp.val)
0072         val_str = utils.helper.mat2str(pp.val);
0073         if length(val_str) > 25
0074             val_str = [val_str(1:25) ' ...'];
0075         end
0076         pstr = [pstr pp.key '=' val_str ' ', ' '];
0077
0078     %%%%%%%%% specwin %%%%%%%%%
0079     elseif isa(pp.val, 'specwin')
0080         % display only the window name here
0081         w = pp.val;
0082         pstr = [pstr pp.key '=' w.type ' ', ' '];
0083
0084     %%%%%%%%% miir %%%%%%%%%
0085     elseif isa(pp.val, 'miir')
0086         f = pp.val;
0087         if strcmp(f.infile, '')
0088             pstr = [pstr pp.key '=' f.name ' ', ' '];
0089         else
0090             pstr = [pstr pp.key '=' f.infile ' ', ' '];
0091         end
0092
0093     %%%%%%%%% mfir %%%%%%%%%
0094     elseif isa(pp.val, 'mfir')
0095         f = pp.val;

```

```
0096     pstr = [pstr pp.key '=' f.name ', '];
0097
0098     %%%%%%%%%% time, timeformat, timespan %%%%%%%%%%
0099     elseif isa(pp.val, 'time') || isa(pp.val, 'timeformat') || isa(pp.val, 'timespan')
0100         pstr = [pstr pp.key '=' char(pp.val) ', '];
0101         %%%%%%%%%% symbols %%%%%%%%%%
0102     elseif isa(pp.val, 'sym')
0103         sval = char(pp.val);
0104         if isempty(sval)
0105             sval = [''];
0106         elseif length(sval) >= 10
0107             sval = [sval(1:10) '...'];
0108         end
0109         pstr = [pstr pp.key '=' sval ', '];
0110         %%%%%%%%%% all other classes %%%%%%%%%%
0111     else
0112         pstr = [pstr pp.key '=' class(pp.val) '-object, '];
0113     end
0114 end
0115
0116 % remove last ', '
0117 if length(pstr)>1
0118     pstr = pstr(1:end-2);
0119 end
0120
0121 %%% Prepare output
0122 varargout{1} = pstr;
0123 end
0124
0125 %%%%%%%%%% Local Functions %%%%%%%%%%
0126 %
0127 %%%%%%%%%%
0128 %%%%%%%%%%
0129 %%%%%%%%%%
0130 %%%%%%%%%%
0131 %
0132 % FUNCTION:      getInfo
0133 %
0134 % DESCRIPTION:   Get Info Object
0135 %
0136 % HISTORY:       11-07-07 M Hewitson
0137 %                Creation.
0138 %
0139 %%%%%%%%%%
0140
0141 function ii = getInfo(varargin)
0142     if nargin == 1 && strcmpi(varargin{1}, 'None')
0143         sets = {};
0144         pl = [];
0145     else
0146         sets = {'Default'};
0147         pl = getDefaultPlist;
0148     end
0149     % Build info object
0150     ii = minfo(mfilename, 'param', '', utils.const.categories.output, '$Id: char.m,v 1.17
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0151 end
0152
0153 %%%%%%%%%%
0154 %
0155 % FUNCTION:      getDefaultPlist
0156 %
0157 % DESCRIPTION:   Get Default Plist
0158 %
0159 % HISTORY:       11-07-07 M Hewitson
0160 %                Creation.
0161 %
0162 %%%%%%%%%%
0163
0164 function plo = getDefaultPlist()
0165     plo = plist();
0166 end
0167
```

display

PURPOSE ^

DISPLAY display a parameter

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY display a parameter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY display a parameter
             Is called for the parameter object when the semicolon is not used.

CALL:       param('a', 1)
             txt = display(param('a', 1));

M-FILE INFO: Get information about this methods by calling
             >> param.getInfo('display')

             Get information about a specified set-plist by calling:
             >> param.getInfo('display', 'None')

VERSION:    $Id: display.m,v 1.36 2008/09/07 10:54:11 hewitson Exp $

HISTORY:    30-01-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a param object into a string.

This function is called by:

-  [display2](#) DISPLAY display a parameter
-  [param](#) PARAM Parameter object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY display a parameter
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY display a parameter
0005 %             Is called for the parameter object when the semicolon is not used.
0006 %
0007 % CALL:       param('a', 1)
0008 %             txt = display(param('a', 1));
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
```

```

0011 %                >> param.getInfo('display')
0012 %
0013 %                Get information about a specified set-plist by calling:
0014 %                >> param.getInfo('display', 'None')
0015 %
0016 % VERSION:      $Id: display.m,v 1.36 2008/09/07 10:54:11 hewitson Exp $
0017 %
0018 % HISTORY:      30-01-2007 M Hewitson
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%
0022 %
0023 function varargout = display(varargin)
0024 %
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 objs = utils.helper.collect_objects(varargin(:), 'param');
0032 %
0033 txt = {};
0034 %
0035 for ii = 1:numel(objs)
0036     banner = sprintf('---- param %d ----', ii);
0037     txt{end+1} = banner;
0038 %
0039 % get key and value
0040 name = objs(ii).key;
0041 v     = objs(ii).val;
0042 %
0043 txt{end+1} = ['key: ' name];
0044 %
0045 % convert value for printing
0046 %
0047 %%% Character %%%
0048 if ischar(v)
0049     MAXLENGTH = 100;
0050     if length(v) > MAXLENGTH
0051         txt{end+1} = ['val: '' v(1:MAXLENGTH) ' ...'];
0052     else
0053         txt{end+1} = ['val: '' v ''];
0054     end
0055 %
0056 %%% Logical %%%
0057 elseif islogical(v)
0058     if v == true
0059         txt{end+1} = 'val: true';
0060     else
0061         txt{end+1} = 'val: false';
0062     end
0063 %
0064 %%% Numeric %%%
0065 elseif isnumeric(v)
0066     sval = '';
0067     if size(v,1) == 1 || size(v,2) == 1
0068         for j=1:min(length(v), 10)
0069             sval = [sval sprintf('%g ', v(j))];
0070         end
0071         if length(v) > 10
0072             sval = [sval '...'];
0073         end
0074     elseif isempty(v)
0075         sval = '[]';
0076     else
0077         sval = sprintf('Matrix [%dx%d]', size(v,1), size(v,2));
0078     end
0079     txt{end+1} = ['val: ' sval];
0080 %
0081 %%% Symbolic Object %%%
0082 elseif isa(v, 'sym')
0083     sval = char(v);
0084     txt{end+1} = ['val: ' sval];
0085 %
0086 %%% Unit Object %%%
0087 elseif isa(v, 'unit')
0088     sval = char(v);
0089     txt{end+1} = ['val: ' sval];
0090 %
0091 %%% Specwin Object %%%
0092 elseif isa(v, 'specwin');
0093     txt{end+1} = ['val: specwin'];
0094     txt = [txt char(v)];
0095 %
0096 %%% Provenance Object %%%

```

```

0097 elseif isa(v, 'provenance')
0098     txt{end+1} = ['val: ' v.creator '@' v.hostname ' (' v.ip ')'];
0099
0100     %%%% miir and mfir Object %%%%
0101 elseif isa(v, 'miir') || isa(v, 'mfir')
0102     txt{end+1} = ['val: ' char(v)];
0103
0104     %%%% time, timespan and timeformat Object %%%%
0105 elseif isa(v, 'time') || isa(v, 'timespan') || isa(v, 'timeformat')
0106     txt{end+1} = ['val: ' char(v)];
0107
0108     %%%% ao %%%%
0109     %%%% cdata, fsdata, tsdata, xydata %%%%
0110     %%%% plist, param %%%%
0111     %%%% history %%%%
0112 elseif isa(v, 'ao') || isa(v, 'fsdata') || isa(v, 'tsdata') || ...
0113     isa(v, 'cdata') || isa(v, 'xydata') || isa(v, 'plist') || ...
0114     isa(v, 'param') || isa(v, 'history')
0115     vinfo = whos('v');
0116     txt{end+1} = ['val: ' vinfo.class '-object'];
0117
0118     %%%% pole, zero and pole/zero model Object %%%%
0119 elseif isa(v, 'pole') || isa(v, 'zero') || isa(v, 'pzmodel')
0120     sval = '';
0121     for j=1:length(v)
0122         sval = [sval ' ' char(v(j))];
0123     end
0124     txt{end+1} = ['val: ' sval];
0125
0126     %%%% Structures %%%%
0127 elseif isstruct(v)
0128
0129     fields = fieldnames(v);
0130     txt{end+1} = 'val: structure';
0131     for ii=1:length(fields)
0132         field = fields{ii};
0133         if ischar(v.(field))
0134             txt{end+1} = [' ' field ': ' v.(field) ''];
0135         elseif isnumeric(v.(field))
0136             txt{end+1} = [' ' field ': ' utils.helper.mat2str(v.(field))];
0137         elseif isobject(v.(field))
0138             txt{end+1} = [' ' field ': ' class(v.(field)) '-object'];
0139         else
0140             txt{end+1} = [' ' field ': ' num2str(size(v.(field),1)) 'x'
num2str(size(v.(field),2)) ' ' class(v.(field))];
0141         end
0142     end
0143
0144     %%%% Cell %%%%
0145 elseif iscell(v)
0146     sval = '{}';
0147     for ii = 1:length(v)
0148         if ischar(v{ii})
0149             sval = [sval ' ' v{ii} ' '];
0150         elseif isnumeric(v{ii})
0151             sval = [sval ' ' utils.helper.mat2str(v{ii}) ' '];
0152         elseif isobject(v{ii})
0153             sval = [sval ' ' class(v{ii}) ' '];
0154         else
0155             error('### Do not use a [%s] in a cell array', class(v));
0156         end
0157     end
0158     if length(sval) > 1
0159         sval(end) = '}';
0160     else
0161         sval(end+1) = '}';
0162     end
0163     txt{end+1} = ['val: ' sval];
0164
0165     %%%% JAVA %%%%
0166 elseif isjava(v)
0167     txt{end+1} = ['val: ' class(v)];
0168
0169 else
0170     if isobject(v)
0171         txt{end+1} = sprintf('val: <%s>', class(v));
0172     elseif isa(v, 'function_handle')
0173         txt{end+1} = sprintf('val: function handle: %s', func2str(v));
0174     else
0175         txt{end+1} = sprintf('val: <unknown type>');
0176     end
0177 end
0178
0179 banner_end(1:length(banner)) = '-';
0180 txt{end+1} = banner_end;
0181

```

```
0182     end
0183
0184     %% Prepare output
0185     if nargout == 0
0186         for ii=1:length(txt)
0187             disp(txt{ii});
0188         end
0189     elseif nargout == 1
0190         varargout{1} = txt;
0191     end
0192
0193 end
0194
0195 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0196 %                                     Local Functions                                     %
0197 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0198
0199 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0200 %
0201 % FUNCTION:      getInfo
0202 %
0203 % DESCRIPTION:  Get Info Object
0204 %
0205 % HISTORY:      11-07-07 M Hewitson
0206 %               Creation.
0207 %
0208 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0209
0210 function ii = getInfo(varargin)
0211     if nargin == 1 && strcmpi(varargin{1}, 'None')
0212         sets = {};
0213         pl   = [];
0214     else
0215         sets = {'Default'};
0216         pl   = getDefaultPlist;
0217     end
0218     % Build info object
0219     ii = minfo(mfilename, 'param', '', utils.const.categories.output, '$Id: display.m,v
1.36 2008/09/07 10:54:11 hewitson Exp $', sets, pl);
0220 end
0221
0222 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0223 %
0224 % FUNCTION:      getDefaultPlist
0225 %
0226 % DESCRIPTION:  Get Default Plist
0227 %
0228 % HISTORY:      11-07-07 M Hewitson
0229 %               Creation.
0230 %
0231 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0232
0233 function plo = getDefaultPlist()
0234     plo = plist();
0235 end
0236
```

display2

PURPOSE ^

DISPLAY display a parameter

SYNOPSIS ^

function varargout = display2(varargin)

DESCRIPTION ^

```
DISPLAY display a parameter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY display a parameter

CALL FOR PARAMETERS:

        pl = obj.getDefaultPlist('display'); Gets the default parameter list
        ver = obj.getVersion('dispaly');    Gets the CVS version string
        cat = obj.getCategory('dispaly');    Gets the category

VERSION:      $Id: display2.m,v 1.4 2008/09/04 15:29:31 ingo Exp $

HISTORY:      30-01-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a param object into a string.
-  [display](#) DISPLAY display a parameter

This function is called by:

SUBFUNCTIONS ^

-  [function txt = main\(txt, prefix, val\)](#)
-  [function txt = atomic_value\(txt, prefix, val, disp_class_only\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY display a parameter
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY display a parameter
0005 %
0006 % CALL FOR PARAMETERS:
0007 %
0008 %         pl = obj.getDefaultPlist('display'); Gets the default parameter list
0009 %         ver = obj.getVersion('dispaly');    Gets the CVS version string
0010 %         cat = obj.getCategory('dispaly');    Gets the category
0011 %
0012 % VERSION:      $Id: display2.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
0013 %
0014 % HISTORY:      30-01-2007 M Hewitson
```



```

0015 %                               Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 function varargout = display2(varargin)
0020
0021 %%% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 objs = utils.helper.collect_objects(varargin(:), 'param');
0028
0029 txt = {};
0030
0031 for i=1:numel(objs)
0032     banner = sprintf('---- param %d ----', i);
0033     txt{end+1} = banner;
0034
0035     % get key and value
0036     name = objs(i).key;
0037     v     = objs(i).val;
0038
0039     txt{end+1} = ['key: ' name];
0040
0041     % convert value for printing
0042     txt = main(txt, 'val: ', v);
0043
0044     banner_end(1:length(banner)) = '-';
0045     txt{end+1} = banner_end;
0046
0047 end
0048
0049 if nargout == 0
0050     char(txt)
0051 end
0052
0053 varargout{1} = txt;
0054
0055 end
0056
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %                               Local Functions
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 function txt = main(txt, prefix, val)
0062
0063 %%% Structures
0064 if isstruct(val)
0065
0066     fields = fieldnames(val);
0067     txt{end+1} = [prefix, 'structure'];
0068     for ii=1:length(fields)
0069         field = fields{ii};
0070         prefix(1:end) = ',';
0071         new_prefix = [prefix field ': '];
0072         txt = main(txt, new_prefix, val.(field));
0073     end
0074
0075 %%% Cell
0076 elseif iscell(val)
0077
0078     if prefix(end) == '[', prefix = prefix(1:end-1); end
0079
0080     txt{end+1} = sprintf('%scell-array {%dx%d}', prefix, size(val,1), size(val,2));
0081     for ii = 1:size(val,1)
0082         for jj = 1:size(val,2)
0083             prefix(1:end) = ',';
0084             if ischar(val{ii,jj})
0085                 new_prefix = sprintf('%s{%d,%d} ', prefix, ii, jj);
0086             else
0087                 new_prefix = sprintf('%s{%d,%d} ', prefix, ii, jj);
0088             end
0089             txt = main(txt, new_prefix, val{ii,jj});
0090         end
0091     end
0092
0093 else
0094     if numel(val) > 1
0095         txt = atomic_value(txt, prefix, val, true);
0096     else
0097         txt = atomic_value(txt, prefix, val, false);
0098     end
0099 end
0100 end

```



```

0101
0102 function txt = atomic_value(txt, prefix, val, disp_class_only)
0103
0104 %%%%%%%%% Character %%%%%%%%%
0105 if ischar(val)
0106     if length(val) > 45
0107         txt{end+1} = [prefix ' ' val(1:45) ' ...'];
0108     else
0109         txt{end+1} = [prefix ' ' val ''];
0110     end
0111
0112 %%%%%%%%% Logical %%%%%%%%%
0113 elseif islogical(val)
0114     if val == true
0115         txt{end+1} = [prefix 'true'];
0116     else
0117         txt{end+1} = [prefix 'false'];
0118     end
0119
0120 %%%%%%%%% Numeric %%%%%%%%%
0121 elseif isnumeric(val)
0122     sval = '';
0123     if size(val,1) == 1 || size(val,2) == 1
0124         for j=1:min(length(val), 10)
0125             sval = [sval sprintf('%g ', val(j))];
0126         end
0127         if length(val) > 10
0128             sval = [sval '...'];
0129         end
0130         if sval(end) == ' '
0131             sval = sval(1:end-1);
0132         end
0133     elseif isempty(val)
0134         if prefix(end) == '[', prefix = prefix(1:end-1); end
0135         sval = '[]';
0136     else
0137         sval = sprintf('matrix [%dx%d]', size(val,1), size(val,2));
0138     end
0139     txt{end+1} = [prefix sval];
0140
0141 %%%%%%%%% Symbolic Object %%%%%%%%%
0142 elseif isa(val, 'sym')
0143     txt{end+1} = [prefix char(val)];
0144
0145 elseif isobject(val)
0146     sval = display(val);
0147
0148     for j=1:numel(sval)
0149         txt{end+1} = [prefix sval{j}];
0150     end
0151
0152 %%%%%%%%% pole, zero and pole/zero model Object %%%%%%%%%
0153 % elseif isa(val, 'pole') || isa(val, 'zero') || isa(val, 'pzmodel')
0154 %     sval = '';
0155 %     for j=1:length(val)
0156 %         sval = [sval char(val(j)) ' '];
0157 %     end
0158 %     sval = sval(1:end-2);
0159 %     txt{end+1} = [prefix sval];
0160 %
0161 % elseif disp_class_only
0162 %     if prefix(end) == '[', prefix = prefix(1:end-1); end
0163 %
0164 %     txt{end+1} = sprintf('%s[%dx%d %s]', prefix, size(val,1), size(val,2),
class(val));
0165 %
0166 % %%%%%%%%% Specwin Object %%%%%%%%%
0167 % elseif isa(val, 'specwin');
0168 %     txt{end+1} = [prefix 'spectral window'];
0169 %     prefix(1:end) = ' ';
0170 %     txt{end+1} = [prefix 'name' = ' val.name'];
0171 %     txt{end+1} = [prefix 'alpha' = ' num2str(val.alpha)'];
0172 %     txt{end+1} = [prefix 'psll' = ' num2str(val.psll)'];
0173 %     txt{end+1} = [prefix 'rov' = ' num2str(val.rov)'];
0174 %     txt{end+1} = [prefix 'nenbw' = ' num2str(val.nenbw)'];
0175 %     txt{end+1} = [prefix 'w3db' = ' num2str(val.w3db)'];
0176 %     txt{end+1} = [prefix 'flatness' = ' num2str(val.flatness)'];
0177 %
0178 % %%%%%%%%% Provenance Object %%%%%%%%%
0179 % elseif isa(val, 'provenance')
0180 %     txt{end+1} = [prefix val.creator '@' val.hostname ' (' val.ip ')'];
0181 %
0182 % %%%%%%%%% miir and mfir Object %%%%%%%%%
0183 % elseif isa(val, 'miir') || isa(val, 'mfir')
0184 %     txt{end+1} = [prefix char(val)];
0185 %

```

```

0186 % %%%% time, timespan and timeformat Object %%%%
0187 % elseif isa(val, 'time') || isa(val, 'timespan') || isa(val, 'timeformat')
0188 %     txt{end+1} = [prefix char(val)];
0189 %
0190 % %%%% ao %%%%
0191 % %%%% cdata, fsdata, tsdata, xydata %%%%
0192 % %%%% plist, param %%%%
0193 % %%%% history %%%%
0194 % elseif isa(val, 'ao') || isa(val, 'fsdata') || isa(val, 'tsdata') || ...
0195 %     isa(val, 'cdata') || isa(val, 'xydata') || isa(val, 'plist') || ...
0196 %     isa(val, 'param') || isa(val, 'history')
0197 %     vinfo = whos('val');
0198 %     txt{end+1} = [prefix vinfo.class '-object'];
0199 %
0200 % %%%% JAVA %%%%
0201 % elseif isjava(val)
0202 %     txt{end+1} = [prefix class(val)];
0203 %
0204 % else
0205 %     if isobject(val)
0206 %         txt{end+1} = sprintf('%s<%s>', prefix, class(val));
0207 %     else
0208 %         txt{end+1} = sprintf('%s<unknown type>', prefix);
0209 %     end
0210 % end
0211 %
0212 % end
0213 %
0214 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0215 %
0216 % FUNCTION:      getInfo
0217 %
0218 % DESCRIPTION:  Get Info Object
0219 %
0220 % HISTORY:      11-07-07 M Hewitson
0221 %               Creation.
0222 %
0223 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0224 %
0225 function ii = getInfo(varargin)
0226     if nargin == 1 && strcmpi(varargin{1}, 'None')
0227         sets = {};
0228         pl = [];
0229     else
0230         sets = {'Default'};
0231         pl = getDefaultPlist;
0232     end
0233     % Build info object
0234     ii = minfo(mfilename, 'param', '', utils.const.categories.output, '$Id: display2.m,v
1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
0235 end
0236 %
0237 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0238 %
0239 % FUNCTION:      getDefaultPlist
0240 %
0241 % DESCRIPTION:  Get Default Plist
0242 %
0243 % HISTORY:      11-07-07 M Hewitson
0244 %               Creation.
0245 %
0246 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0247 %
0248 function plo = getDefaultPlist()
0249     plo = plist();
0250 end
0251 %
0252 %
0253 %
0254 %
0255 %
0256 %
0257 %
0258 %
0259 %
0260 %

```

string

PURPOSE ^

STRING writes a command string that can be used to recreate the input param object.

SYNOPSIS ^

```
function varargout = string(varargin)
```

DESCRIPTION ^

```
STRING writes a command string that can be used to recreate the input param object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING writes a command string that can be used to recreate the
             input param object.

CALL:       cmd = string(param_obj)

INPUT:      param_obj - parameter object

OUTPUT:     cmd       - command string to create the input object

M-FILE INFO: Get information about this methods by calling
              >> param.getInfo('string')

              Get information about a specified set-plist by calling:
              >> param.getInfo('string', 'set')

VERSION:    $Id: string.m,v 1.7 2008/09/04 15:29:31 ingo Exp $

HISTORY:    15-02-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [string](#) STRING writes a command string that can be used to recreate the input param object.

This function is called by:

-  [param](#) PARAM Parameter object class constructor.
-  [string](#) STRING writes a command string that can be used to recreate the input param object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING writes a command string that can be used to recreate the input param object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: STRING writes a command string that can be used to recreate the
0005 %               input param object.
0006 %
0007 % CALL:       cmd = string(param_obj)
```

```

0008 %
0009 % INPUT:      param_obj - parameter object
0010 %
0011 % OUTPUT:     cmd       - command string to create the input object
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> param.getInfo('string')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> param.getInfo('string', 'set')
0018 %
0019 % VERSION:    $Id: string.m,v 1.7 2008/09/04 15:29:31 ingo Exp $
0020 %
0021 % HISTORY:     15-02-2008 Diepholz
0022 %               Creation
0023 %
0024 %%%%%%%%%%%
0025 %
0026 function varargout = string(varargin)
0027 %
0028 % Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 objs = utils.helper.collect_objects(varargin(:), 'param');
0035
0036 cmd = '';
0037
0038 for ii = 1:numel(objs)
0039
0040     key = objs(ii).key;
0041     val = objs(ii).val;
0042
0043     if ischar(val)
0044         val_str = [''' val '''];
0045     elseif isnumeric(val)
0046         val_str = utils.helper.mat2str(val);
0047     elseif isobject(val)
0048         val_str = string(val);
0049     else
0050         error('### Unknown object [%s]', class(val));
0051     end
0052
0053     cmd = [cmd 'param('' key '' , ' val_str ') '];
0054 end
0055
0056 %%% Wrap the command only in bracket if the there are more than one object
0057 if numel(objs) > 1
0058     cmd = ['[' cmd(1:end-1) ']' ];
0059 end
0060
0061 %%% Prepare output
0062 varargout{1} = cmd;
0063 end
0064
0065 %%%%%%%%%%%
0066 %               Local Functions
0067 %%%%%%%%%%%
0068 %%%%%%%%%%%
0069 %%%%%%%%%%%
0070 %
0071 % FUNCTION:     getInfo
0072 %
0073 % DESCRIPTION:  Get Info Object
0074 %
0075 % HISTORY:      11-07-07 M Hewitson
0076 %               Creation.
0077 %
0078 %%%%%%%%%%%
0079 %
0080 function ii = getInfo(varargin)
0081     if nargin == 1 && strcmpi(varargin{1}, 'None')
0082         sets = {};
0083         pl = [];
0084     else
0085         sets = {'Default'};
0086         pl = getDefaultPlist;
0087     end
0088     % Build info object
0089     ii = minfo(mfilename, 'param', '', utils.const.categories.output, '$Id: string.m,v 1.7
2008/09/04 15:29:31 ingo Exp $', sets, pl);
0090 end
0091
0092 %%%%%%%%%%%

```

```
0093 %  
0094 % FUNCTION:      getDefaultPlist  
0095 %  
0096 % DESCRIPTION: Get Default Plist  
0097 %  
0098 % HISTORY:        11-07-07 M Hewitson  
0099 %                  Creation.  
0100 %  
0101 %%%%%%%%%%%%%%%  
0102  
0103 function plo = getDefaultPlist()  
0104     plo = plist();  
0105 end  
0106
```

cdata

PURPOSE ^

CDATA is the constant data class.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
CDATA is the constant data class.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CDATA is the constant data class.

SUPER CLASSES: ltpda_data

SUB CLASSES:

PROPERTIES:

    Protected Properties (read only)
    yunits - units of the y-axis
    y      - a matrix of data samples

CDATA METHODS:

    Defined Abstract methods:
    char          - returns one character string which represents the object
    copy          - copies an object
    display       - displays an object
    update_struct - updates a object structure to the current tbx-version

    Public Methods
    setYunits     - set the property 'yunits'
    setY          - set the property 'y'
    getY          - get the property 'y'
    applymethod   - applies the given method to the input 2D data
    applyoperator - applies the given operator to the two input data objects

M-FILE INFO: The following call returns an minfo object that contains
              information about the cdata constructor:
              >> info = cdata.getInfo
              or  >> info = cdata.getInfo('cdata')

              You can get information about class methods by calling:
              >> info = cdata.getInfo(method)
              e.g. >> info = cdata.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
              >> info = cdata.getInfo(method, set)
              e.g. >> info = cdata.getInfo('cdata', 'Default')

VERSION: $Id: cdata.m,v 1.47 2008/09/07 18:11:30 hewitson Exp $

HISTORY: 09-06-2008 Hewitson
          Creation.

SEE ALSO: ltpda_obj, ao, miir, mfir, specwin, timespan, pzmodel, history, ssm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [cdata](#) CDATA is the constant data class.

-  **char** CHAR convert a cdata-object into a string.
-  **copy** COPY Make copy of cdata objects depending of the second input
-  **getY** GETY Get the property 'y'.
-  **setY** SETY Set the property 'y'.
-  **setYunits** SETYUNITS Set the property 'yunits'.
-  **update_struct** UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  **cdata** CDATA is the constant data class.
-  **copy** COPY Make copy of cdata objects depending of the second input

SUBFUNCTIONS [^]

-  **function** obj = set.yunits(obj, val)
-  **function** obj = set.y(obj, val)
-  **function** obj = cdata(varargin)
-  **function** out = VEROUT()
-  **function** ii = getInfo(varargin)
-  **function** out = SETS()
-  **function** out = getDefaultPlist(set)

SOURCE CODE [^]

```

0001 % CDATA is the constant data class.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CDATA is the constant data class.
0005 %
0006 % SUPER CLASSES: ltpda_data
0007 %
0008 % SUB CLASSES:
0009 %
0010 % PROPERTIES:
0011 %
0012 %     Protected Properties (read only)
0013 %         yunits - units of the y-axis
0014 %         y      - a matrix of data samples
0015 %
0016 % CDATA METHODS:
0017 %
0018 %     Defined Abstract methods:
0019 %         char      - returns one character string which represents the object
0020 %         copy      - copies an object
0021 %         display   - displays an object
0022 %         update_struct - updates a object structure to the current tbx-version
0023 %
0024 %     Public Methods
0025 %         setYunits  - set the property 'yunits'
0026 %         setY       - set the property 'y'
0027 %         getY       - get the property 'y'
0028 %         applymethod - applies the given method to the input 2D data
0029 %         applyoperator - applies the given operator to the two input data objects
0030 %
0031 % M-FILE INFO: The following call returns an minfo object that contains
0032 %               information about the cdata constructor:
0033 %               >> info = cdata.getInfo
0034 %               or   >> info = cdata.getInfo('cdata')
0035 %
0036 %               You can get information about class methods by calling:
0037 %               >> info = cdata.getInfo(method)
0038 %               e.g. >> info = cdata.getInfo('eq')
0039 %
0040 %               You can also restrict the sets of parameters contained in
0041 %               the minfo object by calling:
0042 %               >> info = cdata.getInfo(method, set)
0043 %               e.g. >> info = cdata.getInfo('cdata', 'Default')
0044 %
0045 % VERSION: $Id: cdata.m,v 1.47 2008/09/07 18:11:30 hewitson Exp $
0046 %

```

```

0047 % HISTORY: 09-06-2008 Hewitson
0048 %
0049 %
0050 % SEE ALSO: ltpda_obj, ao, miir, mfir, specwin, timespan, pzmodel, history, ssm
0051 %
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053
0054 classdef cdata < ltpda_data
0055
0056
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %                                     Property definition
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 %----- Public (read/write) Properties -----
0062 properties
0063 end
0064
0065 %----- Protected read-only Properties -----
0066 properties (SetAccess = protected)
0067     yunits = unit;
0068     y       = [];
0069     version = '$Id: cdata.m,v 1.47 2008/09/07 18:11:30 hewitson Exp $';
0070 end
0071
0072 %----- Private Properties -----
0073 properties (GetAccess = protected, SetAccess = protected)
0074 end
0075
0076 %----- Abstract Properties -----
0077 properties (Abstract = true, SetAccess = protected)
0078 end
0079
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %                                     Check property setting
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 methods
0085     function obj = set.yunits(obj, val)
0086         if ischar(val)
0087             obj.yunits = unit(val);
0088         elseif isa(val, 'unit')
0089             obj.yunits = val;
0090         else
0091             error('### The yunits value must be a unit or a string');
0092         end
0093     end
0094     function obj = set.y(obj, val)
0095         if ~isnumeric(val) || ndims(val) ~= 2
0096             error('### The value for the property ''y'' must be a numeric matrix');
0097         end
0098         obj.y = val;
0099     end
0100 end
0101
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103 %                                     Constructor
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105
0106 methods
0107     function obj = cdata(varargin)
0108
0109         %% Call superclass
0110         obj = obj@ltpda_data(varargin{:});
0111
0112         %%% Set default values
0113
0114         if nargin == 0
0115             %%% no inputs
0116             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0117             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0118
0119         elseif nargin == 1
0120             %%% one input
0121             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0123
0124             if isa(varargin{1}, 'cdata')
0125                 %%% Copy ltpda_obj Object
0126                 data = cdata(cdata-object)
0127                 obj = copy(varargin{1}, 1);
0128
0129             elseif isstruct(varargin{1})
0130                 %%% data = cdata(struct)
0131                 %%% Set properties which are declared in this class
0132                 obj.yunits = unit(varargin{1}.yunits);

```



```

0133         obj.y      = varargin{1}.y;
0134         obj.version = varargin{1}.version;
0135
0136         elseif isnumeric(varargin{1})
0137             %%%%%%%%% data = cdata(data- vector, matrix) %%%%%%%%%
0138             obj = obj.setY(varargin{1});
0139         else
0140             error('### Unknown single argument constructor.');
```

```

0141         end
0142
0143         else
0144             error('### Unknown number of arguments.');
```

```

0145         end
0146
0147     end % End constructor
0148
0149 end
0150
0151 %%%%%%%%%
0152 %                                Methods (public)                                %
0153 %%%%%%%%%
0154
0155 methods
0156     varargout = char(varargin)
0157     varargout = copy(varargin)
0158
0159     varargout = setYunits(varargin)
0160     varargout = setY(varargin)
0161     varargout = getY(varargin)
0162 end
0163
0164 %%%%%%%%%
0165 %                                Methods (protected)                                %
0166 %%%%%%%%%
0167
0168 methods (Access = protected)
0169
0170 end
0171
0172 %%%%%%%%%
0173 %                                Methods (public)                                %
0174 %%%%%%%%%
0175
0176 methods
0177 end
0178
0179 %%%%%%%%%
0180 %                                Methods (static)                                %
0181 %%%%%%%%%
0182
0183 methods (Static)
0184     varargout = update_struct(varargin);
0185
0186     function out = VEROUT()
0187         out = '$ID: cdata.m,v 1.47 2008/09/07 18:11:30 hewitson Exp $';
0188     end
0189
0190     function ii = getInfo(varargin)
0191         ii = utils.helper.generic_getInfo(varargin{:}, 'cdata');
0192     end
0193
0194     function out = SETS()
0195         out = {'Default'};
0196     end
0197
0198     function out = getDefaultPlist(set)
0199         switch set
0200             case 'Default'
0201                 out = plist();
0202             otherwise
0203                 error('### Unknown set [%s]', set);
0204         end
0205     end
0206
0207 end
0208
0209 %%%%%%%%%
0210 %                                Methods (abstract)                                %
0211 %%%%%%%%%
0212
0213 methods (Abstract)
0214 end
0215
0216 end
0217

```


applymethod

PURPOSE ^

APPLYMETHOD applies the given method to the input 2D data.

SYNOPSIS ^

function varargout = applymethod(varargin)

DESCRIPTION ^

```
APPLYMETHOD applies the given method to the input 2D data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: APPLYMETHOD applies the given method to the input 2D data.

CALL:
    d = applymethod(d, pl)

INPUTS:
    d      - a cdata object
    pl     - a plist of configuration options

PARAMETERS:
    'method' - the method to apply to the data
    'dim'    - the dimension of the chosen vector to apply the method
               to. This is necessary for functions like mean() when
               applied to matrices held in cdata objects. For tsdata,
               fsdata or xydata, this option has no effect.
               [default: 1]
    'option' - any additional option to pass to the method.

M-FILE INFO: Get information about this methods by calling
    >> cdata.getInfo('applymethod')

             Get information about a specified set-plist by calling:
    >> cdata.getInfo('applymethod', 'None')

VERSION:    $Id: applymethod.m,v 1.4 2008/09/04 15:29:30 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function v = apply(v, method, dim, opt)
-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```
0001 % APPLYMETHOD applies the given method to the input 2D data.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: APPLYMETHOD applies the given method to the input 2D data.
0005 %
0006 % CALL:
0007 %         d = applymethod(d, pl)
0008 %
0009 % INPUTS:
0010 %         d         - a cdata object
0011 %         pl         - a plist of configuration options
0012 %
0013 % PARAMETERS:
0014 %
0015 %         'method' - the method to apply to the data
0016 %         'dim'   - the dimension of the chosen vector to apply the method
0017 %                   to. This is necessary for functions like mean() when
0018 %                   applied to matrices held in cdata objects. For tsdata,
0019 %                   fsdata or xydata, this option has no effect.
0020 %                   [default: 1]
0021 %         'option' - any additional option to pass to the method.
0022 %
0023 % M-FILE INFO: Get information about this methods by calling
0024 %               >> cdata.getInfo('applymethod')
0025 %
0026 %               Get information about a specified set-plist by calling:
0027 %               >> cdata.getInfo('applymethod', 'None')
0028 %
0029 % VERSION:      $Id: applymethod.m,v 1.4 2008/09/04 15:29:30 ingo Exp $
0030 %
0031 % HISTORY:      04-02-2007 M Hewitson
0032 %                Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035 %
0036 function varargout = applymethod(varargin)
0037
0038 %%% Check if this is a call for parameters
0039 if utils.helper.isinfocall(varargin{:})
0040     varargout{1} = getInfo(varargin{3});
0041     return
0042 end
0043
0044 % Get data2D objects and plists
0045 [ds, ds_invars] = utils.helper.collect_objects(varargin(:), 'cdata');
0046 [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist');
0047
0048 % Combine with default plist
0049 pl = combine(pl, getDefaultPlist);
0050
0051 % Get the method to apply
0052 method = find(pl, 'method');
0053 % Get the dimension to operate along
0054 dim = find(pl, 'dim');
0055 % Get any additional option
0056 opt = find(pl, 'option');
0057
0058 % Loop over data objects
0059 for jj=1:numel(ds)
0060     ds(jj).y = apply(ds(jj).y, method, dim, opt);
0061 end
0062
0063 % Set outputs
0064 varargout{1} = ds;
0065
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %                               Local Functions                               %
0070 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0071
0072 %-----
0073 % Apply method to the vector v
0074 %-----
0075 function v = apply(v, method, dim, opt)
0076 if ~isempty(dim) && ~isempty(opt)
0077     % User supplied a dimension and an option
0078     v = feval(method, v, dim, opt);
0079 elseif ~isempty(dim)
0080     % User supplied only a dimension
0081     v = feval(method, v, dim);
0082 elseif ~isempty(opt)
0083     % User supplied only an option
0084     v = feval(method, v, opt);
0085 else
0086     % User supplied only a method
0087     v = feval(method, v);
0088 end
0089 end

```

```
0090
0091 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0092 %
0093 % FUNCTION:      getInfo
0094 %
0095 % DESCRIPTION:  Get Info Object
0096 %
0097 % HISTORY:      11-07-07 M Hewitson
0098 %               Creation.
0099 %
0100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0101
0102 function ii = getInfo(varargin)
0103     if nargin == 1 && strcmpi(varargin{1}, 'None')
0104         sets = {};
0105         pl    = [];
0106     else
0107         sets = {'Default'};
0108         pl    = getDefaultPlist;
0109     end
0110     % Build info object
0111     ii = minfo(mfilename, 'cdata', '', utils.const.categories.internal, '$Id:
applymethod.m,v 1.4 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0112 end
0113
0114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0115 %
0116 % FUNCTION:      getDefaultPlist
0117 %
0118 % DESCRIPTION:  Get Default Plist
0119 %
0120 % HISTORY:      11-07-07 M Hewitson
0121 %               Creation.
0122 %
0123 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0124
0125 %-----
0126 % Default plist
0127 %-----
0128 function pl = getDefaultPlist()
0129     pl = plist('method', '', 'dim', [], 'option', '');
0130 end
0131
```

applyoperator

PURPOSE ^

APPLYOPERATOR applies the given operator to the two input data objects.

SYNOPSIS ^

function varargout = applyoperator(varargin)

DESCRIPTION ^

```
APPLYOPERATOR applies the given operator to the two input data objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: APPLYOPERATOR applies the given operator to the two input
             2D data objects.

CALL:       d = applyoperator(d1, d2, pl)

INPUTS:     d1/2   - an ltpda_data object (cdata, tsdata, fsdata, xydata)
             pl    - a plist of configuration options

PARAMETERS: 'op'   - the operator to apply, e.g. 'plus'

M-FILE INFO: Get information about this methods by calling
              >> cdata.getInfo('applyoperator')

              Get information about a specified set-plist by calling:
              >> cdata.getInfo('applyoperator', 'set')

VERSION:    $Id: applyoperator.m,v 1.8 2008/09/04 15:29:30 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```
0001 % APPLYOPERATOR applies the given operator to the two input data objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: APPLYOPERATOR applies the given operator to the two input
0005 %              2D data objects.
0006 %
0007 % CALL:       d = applyoperator(d1, d2, pl)
0008 %
0009 % INPUTS:     d1/2   - an ltpda_data object (cdata, tsdata, fsdata, xydata)
0010 %              pl    - a plist of configuration options
0011 %
0012 % PARAMETERS: 'op'   - the operator to apply, e.g. 'plus'
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
```

```

0015 %                >> cdata.getInfo('applyoperator')
0016 %
0017 %                Get information about a specified set-plist by calling:
0018 %                >> cdata.getInfo('applyoperator', 'set')
0019 %
0020 % VERSION:      $Id: applyoperator.m,v 1.8 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:      04-02-2007 M Hewitson
0023 %                Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026 %
0027 function varargout = applyoperator(varargin)
0028 %
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034 %
0035 % Get the objects we have in the correct order
0036 objs = [];
0037 for j=1:nargin
0038     if isa(varargin{j}, 'data2D') || isa(varargin{j}, 'cdata')
0039         objs = [objs {varargin{j}}];
0040     end
0041 end
0042 %
0043 % Get data2D objects and plists
0044 [pl] = utils.helper.collect_objects(varargin(:), 'plist');
0045 %
0046 % Combine with default plist
0047 pl = combine(pl, getDefaultPlist);
0048 %
0049 % Get the operator to apply
0050 op = find(pl, 'op');
0051 %
0052 if numel(objs) ~= 2
0053     error('### cdata/applyoperator requires two input data objects to work on.');
```

```

0054 end
0055 %----- Add some rules here.
0056 % cdata
0057 % 1) time-base must match
0058 % 2) y dimensions must match or one must be a single value
0059 %
0060 %
0061 % Decide the type of the output object
0062 if isa(objs{1}, 'data2D')
0063     dout = objs{1};
0064     dout.setY(feval(op, dout.y, objs{2}.y));
0065 elseif isa(objs{2}, 'data2D')
0066     dout = objs{2};
0067     dout.setY(feval(op, objs{1}.y, dout.y));
0068 else
0069     dout = objs{1};
0070     dout.setY(feval(op, dout.y, objs{2}.y));
0071 end
0072 %
0073 varargout{1} = dout;
0074 end
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %                                Local Functions                                %
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 %
0083 % FUNCTION:      getInfo
0084 %
0085 % DESCRIPTION:   Get Info Object
0086 %
0087 % HISTORY:      11-07-07 M Hewitson
0088 %                Creation.
0089 %
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091 %
0092 function ii = getInfo(varargin)
0093 if nargin == 1 && strcmpi(varargin{1}, 'None')
0094     sets = {};
0095     pl = [];
0096 else
0097     sets = {'Default'};
0098     pl = getDefaultPlist;
0099 end
0100 % Build info object

```

```
0101 ii = minfo(mfilename, 'cdata', '', utils.const.categories.internal, '$Id:
applyoperator.m,v 1.8 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0102 end
0103
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %
0106 % FUNCTION:      getDefaultPlist
0107 %
0108 % DESCRIPTION:  Get Default Plist
0109 %
0110 % HISTORY:      11-07-07 M Hewitson
0111 %              Creation.
0112 %
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114
0115 function pl = getDefaultPlist()
0116     pl = plist('op', '');
0117 end
```


char

PURPOSE ^

CHAR convert a cdata-object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a cdata-object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: CHAR convert a cdata object into a string.
CALL:       string = char(fsdata)
M-FILE INFO: Get information about this methods by calling
              >> fsdata.getInfo('char')
              Get information about a specified set-plist by calling:
              >> fsdata.getInfo('char', 'set')
VERSION:    $Id: char.m,v 1.9 2008/09/04 15:29:30 ingo Exp $
HISTORY:    23-07-2007 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a cdata-object into a string.

This function is called by:

-  [cdata](#) CDATA is the constant data class.
-  [char](#) CHAR convert a cdata-object into a string.
-  [display](#) DISPLAY implement terminal display for cdata object.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a cdata-object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a cdata object into a string.
0005 %
0006 % CALL:       string = char(fsdata)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> fsdata.getInfo('char')
0010 %
```

```

0011 %           Get information about a specified set-plist by calling:
0012 %           >> fsdata.getInfo('char', 'set')
0013 %
0014 % VERSION:      $Id: char.m,v 1.9 2008/09/04 15:29:30 ingo Exp $
0015 %
0016 % HISTORY:      23-07-2007 Diepholz
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 % Collect all cdata-objects
0030 objs = utils.helper.collect_objects(varargin(:), '');
0031
0032
0033 %%% Add the size of the data (all)
0034 pstr = sprintf('(Ndata=[%sx%s]', num2str(size(objs.y,1)), num2str(size(objs.y,2)));
0035
0036 %%% Add the sample rate of data (tsdata, fsdata)
0037 if isprop(objs, 'fs')
0038     pstr = sprintf('%s fs=%g,', pstr, objs.fs);
0039 end
0040
0041 %%% Add the length of this time-series in seconds (tsdata)
0042 if isprop(objs, 'nsecs')
0043     pstr = sprintf('%s nsecs=%d,', pstr, objs.nsecs);
0044 end
0045
0046 %%% Add number of averages (fsdata)
0047 if isprop(objs, 'navs')
0048     pstr = sprintf('%s navs=%d,', pstr, objs.navs);
0049 end
0050
0051 %%% Add time-stamp of the first data sample (fsdata, tsdata)
0052 if isprop(objs, 't0')
0053     pstr = sprintf('%s t0=%s,', pstr, char(objs.t0));
0054 end
0055
0056 pstr(end) = ')';
0057
0058 varargout{1} = pstr;
0059 end
0060
0061 %%%%%%%%%%%%%%
0062 %                               Local Functions
0063 %%%%%%%%%%%%%%
0064
0065 %%%%%%%%%%%%%%
0066 %
0067 % FUNCTION:      getInfo
0068 %
0069 % DESCRIPTION:   Get Info Object
0070 %
0071 % HISTORY:      11-07-07 M Hewitson
0072 %                Creation.
0073 %
0074 %%%%%%%%%%%%%%
0075
0076 function ii = getInfo(varargin)
0077     if nargin == 1 && strcmpi(varargin{1}, 'None')
0078         sets = {};
0079         pl    = [];
0080     else
0081         sets = {'Default'};
0082         pl    = getDefaultPlist;
0083     end
0084     % Build info object
0085     ii = minfo(mfilename, 'cdata', '', utils.const.categories.internal, '$Id: char.m,v 1.9
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0086 end
0087
0088 %%%%%%%%%%%%%%
0089 %
0090 % FUNCTION:      getDefaultPlist
0091 %
0092 % DESCRIPTION:   Get Default Plist
0093 %
0094 % HISTORY:      11-07-07 M Hewitson
0095 %                Creation.

```

```
0096 %
0097 %%%%%%%%%%
0098
0099 function plo = getDefaultPlist()
0100     plo = plist();
0101 end
0102
```

copy

PURPOSE ^

COPY Make copy of cdata objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of cdata objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of cdata objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> cdata.getInfo('copy')

VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [cdata](#) CDATA is the constant data class.

This function is called by:

-  [cdata](#) CDATA is the constant data class.
-  [setY](#) SETY Set the property 'y'.
-  [setYunits](#) SETYUNITS Set the property 'yunits'.

SOURCE CODE ^

```
0001 % COPY Make copy of cdata objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of cdata objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> cdata.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('cdata');
0022         s  = size(old);
```

```
0023     obj(s(1),s(2)) = cdata;  
0024     for kk=1:numel(old)  
0025         utils.helper.msg(utils.const.msg.OPROC2, 'copying cdata');  
0026         for jj=1:length(ps)  
0027             obj(kk).(ps{jj}) = old(kk).(ps{jj});  
0028         end  
0029         % reset on copy  
0030         obj(kk).version = cdata.VEROUT;  
0031     end  
0032 else  
0033     obj = old;  
0034 end  
0035 varargout{1} = obj;  
0036 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

getX

PURPOSE ^

GETX Get the property 'x'.

SYNOPSIS ^

function varargout = getX(varargin)

DESCRIPTION ^

```
GETX Get the property 'x'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the property 'x'.

CALL:      val = obj.getX();
           val = obj.getX(idx);
           val = obj.getX(1:10);

INPUTS:    obj - must be a single cdata object.
           idx - index of the data samples

M-FILE INFO: Get information about this methods by calling
              >> fsdata.getInfo('getX')

              Get information about a specified set-plist by calling:
              >> fsdata.getInfo('getX', 'None')

VERSION:   $Id: getX.m,v 1.1 2008/08/04 17:02:02 ingo Exp $

HISTORY:   27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [cdata](#) CData is the constant data class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETX Get the property 'x'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the property 'x'.
0005 %
0006 % CALL:      val = obj.getX();
0007 %           val = obj.getX(idx);
0008 %           val = obj.getX(1:10);
0009 %
0010 % INPUTS:    obj - must be a single cdata object.
0011 %           idx - index of the data samples
0012 %
```

```

0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> fsdata.getInfo('getX')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> fsdata.getInfo('getX', 'None')
0018 %
0019 % VERSION: $Id: getX.m,v 1.1 2008/08/04 17:02:02 ingo Exp $
0020 %
0021 % HISTORY: 27-05-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025 %
0026 function varargout = getX(varargin)
0027 %
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033 %
0034 %%% get idx
0035 idx = [];
0036 if nargin == 2
0037     if isnumeric(varargin{2})
0038         idx = varargin{2};
0039     else
0040         error('### Unknown arguments to getX.');
```

0041 end

0042 end

0043

0044 x = varargin{1}.x;

0045

0046 %%% get x values

0047 if isempty(idx)

0048 varargout{1} = x;

0049 else

0050 if isempty(x)

0051 x = cell(size(varargin{1}.y));

0052 for ii=1:numel(x), x{ii} = ''; end

0053 end

0054 varargout{1} = x(idx);

0055 end

0056 end

0057

0058 %%

0059 % Local Functions %

0060 %%

0061 %

0062 %%

0063 %

0064 % FUNCTION: getInfo

0065 %

0066 % DESCRIPTION: Get Info Object

0067 %

0068 % HISTORY: 11-07-07 M Hewitson

0069 % Creation.

0070 %

0071 %%

0072 %

0073 function ii = getInfo(varargin)

0074 if nargin == 1 && strcmpi(varargin{1}, 'None')

0075 sets = {};

0076 pl = [];

0077 else

0078 sets = {'Default'};

0079 pl = getDefaultPlist;

0080 end

0081 % Build info object

0082 ii = minfo(mfilename, 'cdata', '', 'Internal', '\$Id: getX.m,v 1.1 2008/08/04 17:02:02

ingo Exp \$', sets, pl);

0083 end

0084

0085 %%

0086 %

0087 % FUNCTION: getDefaultPlist

0088 %

0089 % DESCRIPTION: Get Default Plist

0090 %

0091 % HISTORY: 11-07-07 M Hewitson

0092 % Creation.

0093 %

0094 %%

0095 %

0096 function plo = getDefaultPlist()

0097 plo = plist();

0098 end

0099

getY

PURPOSE ^

GETY Get the property 'y'.

SYNOPSIS ^

function varargout = getY(varargin)

DESCRIPTION ^

```
GETY Get the property 'y'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the property 'y'.

CALL:      val = obj.getY();
           val = obj.getY(idx);
           val = obj.getY(1:10);

INPUTS:    obj - must be a single cdata object.
           idx - index of the data samples

M-FILE INFO: Get information about this methods by calling
             >> fsdata.getInfo('getY')

             Get information about a specified set-plist by calling:
             >> fsdata.getInfo('getY', 'None')

VERSION:   $Id: getY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:   27-05-2008 Diepholz
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [cdata](#) CDATA is the constant data class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETY Get the property 'y'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the property 'y'.
0005 %
0006 % CALL:      val = obj.getY();
0007 %           val = obj.getY(idx);
0008 %           val = obj.getY(1:10);
0009 %
0010 % INPUTS:    obj - must be a single cdata object.
0011 %           idx - index of the data samples
0012 %
```

```
0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> fsdata.getInfo('getY')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> fsdata.getInfo('getY', 'None')
0018 %
0019 % VERSION: $Id: getY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY: 27-05-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = getY(varargin)
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 %%% get idx
0035 idx = [];
0036 if nargin == 2
0037     if isnumeric(varargin{2})
0038         idx = varargin{2};
0039     else
0040         error('### Unknown arguments to getY.');
```

end

%% get y values

```
if isempty(idx)
    varargout{1} = varargin{1}.y;
else
    varargout{1} = varargin{1}.y(idx);
end
end
```

%% Local Functions %

```
% FUNCTION: getInfo
% DESCRIPTION: Get Info Object
% HISTORY: 11-07-07 M Hewitson
% Creation.
```

```
function ii = getInfo(varargin)
    if nargin == 1 && strcmpi(varargin{1}, 'None')
        sets = {};
        pl = [];
    else
        sets = {'Default'};
        pl = getDefaultPlist;
    end
    % Build info object
    ii = minfo(mfilename, 'cdata', '', utils.const.categories.internal, '$Id: getY.m,v 1.2
2008/09/04 15:29:30 ingo Exp $', sets, pl);
end
```

```
% FUNCTION: getDefaultPlist
% DESCRIPTION: Get Default Plist
% HISTORY: 11-07-07 M Hewitson
% Creation.
```

```
function plo = getDefaultPlist()
    plo = plist();
end
```

setX

PURPOSE ^

SETX Set the property 'x'.

SYNOPSIS ^

function varargout = setX(varargin)

DESCRIPTION ^

```
SETX Set the property 'x'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'x'.

CALL:
           obj.setX([1 2 3]);
           obj = obj.setX([1 2 3]); create copy of the object

INPUTS:
           obj - must be a single cdata object.

M-FILE INFO: Get information about this methods by calling
               >> fsdata.getInfo('setX')

               Get information about a specified set-plist by calling:
               >> fsdata.getInfo('setX', 'None')

VERSION:
           $Id: setX.m,v 1.1 2008/08/04 17:02:02 ingo Exp $

HISTORY:
           27-05-2008 Diepholz
                   Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of cdata objects depending of the second input

This function is called by:

 [cdata](#) CDATA is the constant data class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETX Set the property 'x'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'x'.
0005 %
0006 % CALL:
0007 %           obj.setX([1 2 3]);
0008 %           obj = obj.setX([1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %           obj - must be a single cdata object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
```

```
0012 %                >> fsdata.getInfo('setX')
0013 %
0014 %                Get information about a specified set-plist by calling:
0015 %                >> fsdata.getInfo('setX', 'None')
0016 %
0017 % VERSION:      $Id: setX.m,v 1.1 2008/08/04 17:02:02 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setX(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'x'
0039 obj.x = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                Local Functions
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'cdata', '', 'Internal', '$Id: setX.m,v 1.1 2008/08/04 17:02:02
ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('x', '');
0084 end
0085
```

setXY

PURPOSE ^

SETXY Set the property 'xy'.

SYNOPSIS ^

function varargout = setXY(varargin)

DESCRIPTION ^

```
SETXY Set the property 'xy'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'xy'.

CALL:
    obj = obj.setXY([1 2 3], [1 2 3]);
    obj = obj.setXY([1 2 3], [1 2 3]); create copy of the object

INPUTS:
    obj - must be a single cdata object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setXY')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setXY', 'None')

VERSION:
    $Id: setXY.m,v 1.1 2008/08/04 17:02:02 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of cdata objects depending of the second input

This function is called by:

 [cdata](#) CDATA is the constant data class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETXY Set the property 'xy'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'xy'.
0005 %
0006 % CALL:
0007 %     obj = obj.setXY([1 2 3], [1 2 3]);
0008 %     obj = obj.setXY([1 2 3], [1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single cdata object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
```

```
0012 %           >> fsdata.getInfo('setXY')
0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> fsdata.getInfo('setXY', 'None')
0016 %
0017 % VERSION:      $Id: setXY.m,v 1.1 2008/08/04 17:02:02 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setXY(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 x   = varargin{2};
0034 y   = varargin{3};
0035 %
0036 %%% decide whether we modify the pz-object, or create a new one.
0037 obj = copy(obj, nargout);
0038 %
0039 %%% set 'x' and 'y'
0040 obj.x = x;
0041 obj.y = y;
0042 %
0043 varargout{1} = obj;
0044 end
0045 %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %                                Local Functions                                %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 % FUNCTION:      getInfo
0053 %
0054 % DESCRIPTION:   Get Info Object
0055 %
0056 % HISTORY:      11-07-07 M Hewitson
0057 %                Creation.
0058 %
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060 %
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl   = [];
0065     else
0066         sets = {'Default'};
0067         pl   = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'cdata', '', 'Internal', '$Id: setXY.m,v 1.1 2008/08/04 17:02:02
ingo Exp $', sets, pl);
0071 end
0072 %
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getDefaultPlist
0076 %
0077 % DESCRIPTION:   Get Default Plist
0078 %
0079 % HISTORY:      11-07-07 M Hewitson
0080 %                Creation.
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083 %
0084 function plo = getDefaultPlist()
0085     plo = plist('xy', '');
0086 end
0087 %
```

setXunits

PURPOSE ^

SETXUNITS Set the property 'xunits'.

SYNOPSIS ^

function varargout = setXunits(varargin)

DESCRIPTION ^

```
SETXUNITS Set the property 'xunits'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'xunits'.

CALL:
    obj.setXunits('my units');
    obj = obj.setXunits('my units'); create copy of the object

INPUTS:
    obj - must be a single cdata object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setXunits')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setXunits', 'None')

VERSION:
    $Id: setXunits.m,v 1.2 2008/08/15 11:45:19 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of cdata objects depending of the second input

This function is called by:

 [cdata](#) CDATA is the constant data class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETXUNITS Set the property 'xunits'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'xunits'.
0005 %
0006 % CALL:
0007 %     obj.setXunits('my units');
0008 %     obj = obj.setXunits('my units'); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single cdata object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setXunits')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 %     >> fsdata.getInfo('setXunits', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setXunits.m,v 1.2 2008/08/15 11:45:19 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0012 %                >> fsdata.getInfo('setXunits')
0013 %
0014 %                Get information about a specified set-plist by calling:
0015 %                >> fsdata.getInfo('setXunits', 'None')
0016 %
0017 % VERSION:      $Id: setXunits.m,v 1.2 2008/08/15 11:45:19 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setXunits(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'xunits'
0039 if ischar(val)
0040     obj.xunits = sym(val);
0041 else
0042     obj.xunits = val;
0043 end
0044 %
0045 varargout{1} = obj;
0046 end
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %                Local Functions
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053 %
0054 % FUNCTION:      getInfo
0055 %
0056 % DESCRIPTION:   Get Info Object
0057 %
0058 % HISTORY:      11-07-07 M Hewitson
0059 %                Creation.
0060 %
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062 %
0063 function ii = getInfo(varargin)
0064 if nargin == 1 && strcmpi(varargin{1}, 'None')
0065     sets = {};
0066     pl = [];
0067 else
0068     sets = {'Default'};
0069     pl = getDefaultPlist;
0070 end
0071 % Build info object
0072 ii = minfo(mfilename, 'cdata', '', 'Internal', '$Id: setXunits.m,v 1.2 2008/08/15
11:45:19 ingo Exp $', sets, pl);
0073 end
0074 %
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076 %
0077 % FUNCTION:      getDefaultPlist
0078 %
0079 % DESCRIPTION:   Get Default Plist
0080 %
0081 % HISTORY:      11-07-07 M Hewitson
0082 %                Creation.
0083 %
0084 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0085 %
0086 function plo = getDefaultPlist()
0087 plo = plist('xunits', '');
0088 end
0089 %

```


setY

PURPOSE ^

SETY Set the property 'y'.

SYNOPSIS ^

function varargout = setY(varargin)

DESCRIPTION ^

```
SETY Set the property 'y'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'y'.

CALL:
           obj.setY([1 2 3]);
           obj = obj.setY([1 2 3]); create copy of the object

INPUTS:
           obj - must be a single cdata object.

M-FILE INFO: Get information about this methods by calling
               >> fsdata.getInfo('setY')

               Get information about a specified set-plist by calling:
               >> fsdata.getInfo('setY', 'None')

VERSION:
           $Id: setY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
           27-05-2008 Diepholz
                   Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of cdata objects depending of the second input

This function is called by:

 [cdata](#) CDATA is the constant data class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETY Set the property 'y'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'y'.
0005 %
0006 % CALL:
0007 %           obj.setY([1 2 3]);
0008 %           obj = obj.setY([1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %           obj - must be a single cdata object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %               >> fsdata.getInfo('setY')
0015 %
0016 %               Get information about a specified set-plist by calling:
0017 %               >> fsdata.getInfo('setY', 'None')
0018 %
0019 % VERSION:
0020 %           $Id: setY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:
0023 %           27-05-2008 Diepholz
0024 %                   Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %                >> fsdata.getInfo('setY')
0013 %
0014 %                Get information about a specified set-plist by calling:
0015 %                >> fsdata.getInfo('setY', 'None')
0016 %
0017 % VERSION:      $Id: setY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setY(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'y'
0039 obj.y = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                Local Functions
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'cdata', '', utils.const.categories.internal, '$Id: setY.m,v 1.2
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('y', '');
0084 end
0085
```

setYunits

PURPOSE ^

SETYUNITS Set the property 'yunits'.

SYNOPSIS ^

function varargout = setYunits(varargin)

DESCRIPTION ^

```
SETYUNITS Set the property 'yunits'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'yunits'.

CALL:
    obj.setYunits('my units');
    obj = obj.setYunits('my units'); create copy of the object

INPUTS:
    obj - must be a single cdata object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setYunits')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setYunits', 'None')

VERSION:
    $Id: setYunits.m,v 1.4 2008/09/05 14:16:30 hewitson Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of cdata objects depending of the second input

This function is called by:

 [cdata](#) CDATA is the constant data class.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETYUNITS Set the property 'yunits'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'yunits'.
0005 %
0006 % CALL:
0007 %     obj.setYunits('my units');
0008 %     obj = obj.setYunits('my units'); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single cdata object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setYunits')
0015 %
0016 %     Get information about a specified set-plist by calling:
0017 %     >> fsdata.getInfo('setYunits', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setYunits.m,v 1.4 2008/09/05 14:16:30 hewitson Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0012 %                >> fsdata.getInfo('setYunits')
0013 %
0014 %                Get information about a specified set-plist by calling:
0015 %                >> fsdata.getInfo('setYunits', 'None')
0016 %
0017 % VERSION:      $Id: setYunits.m,v 1.4 2008/09/05 14:16:30 hewitson Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setYunits(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'yunits'
0039 if ischar(val)
0040     obj.yunits = unit(val);
0041 else
0042     obj.yunits = val;
0043 end
0044 %
0045 varargout{1} = obj;
0046 end
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %                Local Functions
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053 %
0054 % FUNCTION:      getInfo
0055 %
0056 % DESCRIPTION:   Get Info Object
0057 %
0058 % HISTORY:      11-07-07 M Hewitson
0059 %                Creation.
0060 %
0061 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0062 %
0063 function ii = getInfo(varargin)
0064 if nargin == 1 && strcmpi(varargin{1}, 'None')
0065     sets = {};
0066     pl = [];
0067 else
0068     sets = {'Default'};
0069     pl = getDefaultPlist;
0070 end
0071 % Build info object
0072 ii = minfo(mfilename, 'cdata', '', utils.const.categories.internal, '$Id:
setYunits.m,v 1.4 2008/09/05 14:16:30 hewitson Exp $', sets, pl);
0073 end
0074 %
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076 %
0077 % FUNCTION:      getDefaultPlist
0078 %
0079 % DESCRIPTION:   Get Default Plist
0080 %
0081 % HISTORY:      11-07-07 M Hewitson
0082 %                Creation.
0083 %
0084 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0085 %
0086 function plo = getDefaultPlist()
0087 plo = plist('yunits', '');
0088 end
0089

```

display

PURPOSE ^

DISPLAY implement terminal display for cdata object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY implement terminal display for cdata object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY implement terminal display for cdata object.

M-FILE INFO: Get information about this methods by calling
              >> cdata.getInfo('display')

              Get information about a specified set-plist by calling:
              >> cdata.getInfo('display', 'None')

VERSION:      $Id: display.m,v 1.14 2008/09/05 14:17:51 hewitson Exp $

HISTORY:      31-01-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a cdata-object into a string.

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY implement terminal display for cdata object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY implement terminal display for cdata object.
0005 %
0006 % M-FILE INFO: Get information about this methods by calling
0007 %              >> cdata.getInfo('display')
0008 %
0009 %              Get information about a specified set-plist by calling:
0010 %              >> cdata.getInfo('display', 'None')
0011 %
0012 % VERSION:      $Id: display.m,v 1.14 2008/09/05 14:17:51 hewitson Exp $
0013 %
0014 % HISTORY:      31-01-2007 M Hewitson
0015 %              Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 function varargout = display(varargin)
```

```

0020
0021 %% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 cdatas = utils.helper.collect_objects(varargin(:), 'cdata');
0028
0029 txt = {};
0030
0031 for i=1:numel(cdatas)
0032     banner = sprintf('----- cdata %02d -----', i);
0033     txt{end+1} = banner;
0034
0035     y = cdatas(i).y;
0036     yinfo = whos('y');
0037
0038     txt{end+1} = sprintf('      y: [%dx%d], %s', yinfo.size(1), yinfo.size(2),
yinfo.class);
0039     txt{end+1} = sprintf('yunits: %s', char(cdatas(i).yunits));
0040
0041     banner_end(1:length(banner)) = '-';
0042     txt{end+1} = banner_end;
0043
0044     txt{end+1} = ' ';
0045 end
0046
0047 if nargin == 0
0048     for ii=1:length(txt)
0049         disp(txt{ii});
0050     end
0051 end
0052
0053 varargout{1} = txt;
0054 end
0055
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057 % Local Functions
0058 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060 %
0061 % FUNCTION:      getInfo
0062 %
0063 % DESCRIPTION:  Get Info Object
0064 %
0065 % HISTORY:      11-07-07 M Hewitson
0066 %               Creation.
0067 %
0068 %
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070
0071 function ii = getInfo(varargin)
0072     if nargin == 1 && strcmpi(varargin{1}, 'None')
0073         sets = {};
0074         pl = [];
0075     else
0076         sets = {'Default'};
0077         pl = getDefaultPlist;
0078     end
0079     % Build info object
0080     ii = minfo(mfilename, 'cdata', '', utils.const.categories.output, '$Id: display.m,v
1.14 2008/09/05 14:17:51 hewitson Exp $', sets, pl);
0081 end
0082
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084 %
0085 % FUNCTION:      getDefaultPlist
0086 %
0087 % DESCRIPTION:  Get Default Plist
0088 %
0089 % HISTORY:      11-07-07 M Hewitson
0090 %               Creation.
0091 %
0092 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0093
0094 function plo = getDefaultPlist()
0095     plo = plist();
0096 end
0097

```

fsdata

PURPOSE ^

FSDATA frequency-series object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
FSDATA frequency-series object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FSDATA frequency-series object class constructor.
             Create a frequency-series data object.

SUPER CLASSES: data2D < ltpda_data < ltpda_nuo < ltpda_obj

PROPERTIES:

  Inherit Properties (read only)
    version - cvs-version string
    xunits  - units of the y-axis
    yunits  - units of the y-axis
    x       - data values of the x-axis
    y       - data values of the y-axis

  Properties (read only)
    enbw    - equivalent noise bandwidth
    navs    - number of averages
    fs      - sample rate of data
    t0      - time-stamp of the first data sample

CONSTRUCTORS:

  fsd = fsdata()           - creates a blank frequency-series object
  fsd = fsdata(y)          - creates a frequency-series object with the given
                           y-data. Sample rate of the data is assumed to
                           be 1Hz.
  fsd = fsdata(f,y)        - creates a frequency-series object with the given
                           (x,y)-data. The sample rate is then set as
                           2*x(end).
  fsd = fsdata(y,fs)       - creates a frequency-series object with the given
                           y-data and sample rate. The frequency
                           vector is grown assuming the first y
                           sample corresponds to 0Hz and the last
                           sample corresponds to the Nyquist
                           frequency.
  fsd = fsdata(x,y,fs)     - creates a frequency-series object with the given
                           x,y-data and sample rate.

FSDATA METHODS:

  Defined Abstract methods:
    char      - returns one character string which represents the object
    copy      - copies an object
    display   - displays an object
    update_struct - updates a object structure to the current tbx-version

  Public methods:
    setT0     - set the property 't0'
    setFs     - set the property 'fs'
    setNavs   - set the property 'navs'
    setEnbw   - set the property 'enbw'

M-FILE INFO:  The following call returns an minfo object that contains
               information about the fsdata constructor:
               >> info = fsdata.getInfo
               or  >> info = fsdata.getInfo('fsdata')

               You can get information about class methods by calling:
               >> info = fsdata.getInfo(method)
```

```
e.g. >> info = fsdata.getInfo('eq')

You can also restrict the sets of parameters contained in
the minfo object by calling:
    >> info = fsdata.getInfo(method, set)
e.g. >> info = fsdata.getInfo('fsdata', 'Default')

VERSION:  $Id: fsdata.m,v 1.39 2008/09/03 16:32:18 hewitson Exp $

HISTORY:   30-01-2007 Hewitson
           Creation

SEE ALSO:  tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of fsdata objects depending of the second input
-  [display](#) DISPLAY implement terminal display for fsdata object.
-  [fsdata](#) FSDATA frequency-series object class constructor.
-  [getFfromYFs](#) GETDSFROMYFS grows an evenly spaced frequency vector of N points for
-  [setEnbw](#) SETENBW Set the property 'enbw'.
-  [setFs](#) SETFS Set the property 'fs'.
-  [setNavs](#) SETNAVS Set the property 'navs'.
-  [setT0](#) SETT0 Set the property 't0'.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [copy](#) COPY Make copy of fsdata objects depending of the second input
-  [fsdata](#) FSDATA frequency-series object class constructor.

SUBFUNCTIONS ^

-  [function obj = set.t0\(obj, val\)](#)
-  [function obj = set.navs\(obj, val\)](#)
-  [function obj = set.fs\(obj, val\)](#)
-  [function obj = set.enbw\(obj, val\)](#)
-  [function obj = fsdata\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 % FSDATA frequency-series object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FSDATA frequency-series object class constructor.
0005 %              Create a frequency-series data object.
0006 %
0007 % SUPER CLASSES: data2D < ltpda_data < ltpda_nuo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         version - cvs-version string
0013 %         xunits  - units of the y-axis
0014 %         yunits  - units of the y-axis
0015 %         x       - data values of the x-axis
```



```

0016 %      y      - data values of the y-axis
0017 %
0018 %      Properties (read only)
0019 %      enbw    - equivalent noise bandwidth
0020 %      navs    - number of averages
0021 %      fs      - sample rate of data
0022 %      t0      - time-stamp of the first data sample
0023 %
0024 % CONSTRUCTORS:
0025 %
0026 %      fsd = fsdata()      - creates a blank frequency-series object
0027 %      fsd = fsdata(y)    - creates a frequency-series object with the given
0028 %                          y-data. Sample rate of the data is assumed to
0029 %                          be 1Hz.
0030 %      fsd = fsdata(f,y)  - creates a frequency-series object with the given
0031 %                          (x,y)-data. The sample rate is then set as
0032 %                          2*x(end).
0033 %      fsd = fsdata(y,fs) - creates a frequency-series object with the given
0034 %                          y-data and sample rate. The frequency
0035 %                          vector is grown assuming the first y
0036 %                          sample corresponds to 0Hz and the last
0037 %                          sample corresponds to the Nyquist
0038 %                          frequency.
0039 %      fsd = fsdata(x,y,fs) - creates a frequency-series object with the given
0040 %                          x,y-data and sample rate.
0041 %
0042 % FSDATA METHODS:
0043 %
0044 %      Defined Abstract methods:
0045 %      char      - returns one character string which represents the object
0046 %      copy      - copies an object
0047 %      display   - displays an object
0048 %      update_struct - updates a object structure to the current tbx-version
0049 %
0050 %      Public methods:
0051 %      setT0     - set the property 't0'
0052 %      setFs     - set the property 'fs'
0053 %      setNavs   - set the property 'navs'
0054 %      setEnbw   - set the property 'enbw'
0055 %
0056 % M-FILE INFO: The following call returns an minfo object that contains
0057 %              information about the fsdata constructor:
0058 %              >> info = fsdata.getInfo
0059 %              or  >> info = fsdata.getInfo('fsdata')
0060 %
0061 %              You can get information about class methods by calling:
0062 %              >> info = fsdata.getInfo(method)
0063 %              e.g. >> info = fsdata.getInfo('eq')
0064 %
0065 %              You can also restrict the sets of parameters contained in
0066 %              the minfo object by calling:
0067 %              >> info = fsdata.getInfo(method, set)
0068 %              e.g. >> info = fsdata.getInfo('fsdata', 'Default')
0069 %
0070 % VERSION:  $Id: fsdata.m,v 1.39 2008/09/03 16:32:18 hewitson Exp $
0071 %
0072 % HISTORY:  30-01-2007 Hewitson
0073 %              Creation
0074 %
0075 % SEE ALSO: tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %
0079 % classdef fsdata < data2D
0080 %
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %      Property definition
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0084 %
0085 %----- Public (read/write) Properties -----
0086 % properties
0087 % end
0088 %
0089 %----- Protected read-only Properties -----
0090 % properties (SetAccess = protected)
0091 %     t0      = time(0);
0092 %     navs    = NaN;
0093 %     fs      = NaN;
0094 %     enbw    = NaN;
0095 %     version = '$Id: fsdata.m,v 1.39 2008/09/03 16:32:18 hewitson Exp $';
0096 % end
0097 %
0098 %----- Private Properties -----
0099 % properties (GetAccess = protected, SetAccess = protected)
0100 % end
0101 %

```

```

0102 %----- Abstract Properties -----
0103 properties (Abstract = true, SetAccess = protected)
0104 end
0105
0106 %%%%%%%%%%%%%%% Check property setting %%%%%%%%%%%%%%%
0107 %
0108 %%%%%%%%%%%%%%%
0109
0110 methods
0111 function obj = set.t0(obj, val)
0112     if ~isa(val, 'time') || isempty(val)
0113         error('### The value for the property ''t0'' must be a time object');
0114     end
0115     obj.t0 = val;
0116 end
0117 function obj = set.navs(obj, val)
0118     if ~isnumeric(val) || isempty(val) || length(val) < 0 || (~isnan(val) &&
0119 rem(val,1)~=0)
0119         error('### The value for the property ''navs'' must be a positive integer');
0120     end
0121     obj.navs = val;
0122 end
0123 function obj = set.fs(obj, val)
0124     if ~isempty(val)
0125         if ~isnumeric(val) || ~isreal(val) || val < 0
0126             error('### The value for the property ''fs'' must be a real positive
0127 number');
0128         end
0129     end
0130     obj.fs = val;
0131 end
0132 function obj = set.enbw(obj, val)
0133     if ~isnumeric(val) || ~isreal(val) || any(val < 0)
0134         error('### The value for the property ''enbw'' must be a real positive number
0135 or a vector');
0136     end
0137     if ~isempty(val) && ~isempty(obj.y)
0138         if length(val) ~=1 && (length(val) ~= length(obj.y))
0139             error('### The ENBW can only be a single number, of a vector the same length
0140 as the y data. ');
0141         end
0142     end
0143     obj.enbw = val;
0144 end
0145 %%%%%%%%%%%%%%% Constructor %%%%%%%%%%%%%%%
0146 %
0147 %%%%%%%%%%%%%%%
0148
0149 methods
0150 function obj = fsdata(varargin)
0151
0152     %% Call superclass
0153     obj = obj@data2D(varargin{:});
0154
0155     %%%%%%%%%%% Set default values %%%%%%%%%%%
0156
0157     if nargin == 0
0158         %%%%%%%%%%% no inputs %%%%%%%%%%%
0159         %%%%%%%%%%%
0160
0161     elseif nargin == 1
0162         %%%%%%%%%%% one input %%%%%%%%%%%
0163         %%%%%%%%%%%
0164         %%%%%%%%%%%
0165
0166         if isa(varargin{1}, 'fsdata')
0167             %%%%%%%%%%% data = fsdata(fsdata-object) %%%%%%%%%%%
0168             %----- Copy fsdata Object
0169             obj = copy(varargin{1}, 1);
0170
0171         elseif isstruct(varargin{1})
0172             %%%%%%%%%%% data = fsdata(struct) %%%%%%%%%%%
0173             %----- struct
0174             obj.t0 = utils.helper.struct2obj(varargin{1}.t0, 'time');
0175             obj.navs = varargin{1}.navs;
0176             obj.fs = varargin{1}.fs;
0177             obj.enbw = varargin{1}.enbw;
0178             obj.version = varargin{1}.version;
0179
0180         elseif isnumeric(varargin{1})
0181             %%%%%%%%%%% data = fsdata(y-vector) %%%%%%%%%%%
0182             %----- y vector
0183             obj.setY(varargin{1});

```

```

0184         obj.setFs(1);
0185         obj.setX(fsdata.getFfromYFs(length(obj.y), obj.fs));
0186
0187     else
0188         error('### Unknown single argument constructor.');
```

end

```

0189
0190 elseif nargin == 2
0191     %%%%%%%%%%% two input %%%%%%%%%%%
0192     %%%%%%%%%%% two input %%%%%%%%%%%
0193     %%%%%%%%%%% two input %%%%%%%%%%%
0194
0195     if numel(varargin{1}) > numel(varargin{2}) && numel(varargin{2}) == 1
0196         %%%%%%%%%%% data = fsdata(y-vector, fs) %%%%%%%%%%%
0197         % tsdata(y,fs)
0198         obj.setY(varargin{1});
0199         obj.setFs(varargin{2});
0200         obj.setX(fsdata.getFfromYFs(length(obj.y), obj.fs));
0201
0202     elseif numel(varargin{1}) == numel(varargin{2})
0203         %%%%%%%%%%% data = fsdata(x-vector, y-vector) %%%%%%%%%%%
0204         % tsdata(x,y)
0205         obj.setX(varargin{1});
0206         obj.setY(varargin{2});
0207
0208     else
0209         error('### Unknown two argument constructor.');
```

end

```

0210
0211 elseif nargin == 3
0212     %%%%%%%%%%% three input %%%%%%%%%%%
0213     %%%%%%%%%%% three input %%%%%%%%%%%
0214     %%%%%%%%%%% three input %%%%%%%%%%%
0215
0216     if numel(varargin{1}) == numel(varargin{2}) && numel(varargin{3}) == 1
0217         %%%%%%%%%%% data = fsdata(x-vector, y-vector, fs) %%%%%%%%%%%
0218         % tsdata(x,y,fs)
0219         obj.setXY(varargin{1}, varargin{2});
0220         obj.setFs(varargin{3});
0221
0222     else
0223         error('### Unknown three argument constructor.');
```

end

```

0224
0225 else
0226     error('### Unknown number of constructor arguments.');
```

end

```

0227
0228 end % End constructor
0229 end % End public methods
0230
0231 %%%%%%%%%%% Methods (Public) %%%%%%%%%%%
0232 %
0233 %%%%%%%%%%% Methods (Public) %%%%%%%%%%%
0234
0235 methods
0236 % Setters
0237     varargout = setT0(varargin);
0238     varargout = setFs(varargin);
0239     varargout = setNavs(varargin);
0240     varargout = setEnbw(varargin);
0241
0242 % Other public methods
0243     varargout = copy(varargin);
0244     varargout = display(varargin);
0245 end % End public methods
0246
0247 %%%%%%%%%%% Methods (protected) %%%%%%%%%%%
0248 %
0249 %%%%%%%%%%% Methods (protected) %%%%%%%%%%%
0250
0251 methods (Access = protected)
0252 end
0253
0254 %%%%%%%%%%% Methods (private) %%%%%%%%%%%
0255 %
0256 %%%%%%%%%%% Methods (private) %%%%%%%%%%%
0257
0258 methods (Access = private)
0259
0260 end
0261 %%%%%%%%%%% Methods (static) %%%%%%%%%%%
0262 %
0263 %%%%%%%%%%% Methods (static) %%%%%%%%%%%
0264 methods (Static)
0265     varargout = update_struct(varargin);
0266
0267 function out = VEROUT()
0268     out = '$Id: fsdata.m,v 1.39 2008/09/03 16:32:18 hewitson Exp $';
0269 end
```

```
0270
0271     function ii = getInfo(varargin)
0272         ii = utils.helper.generic_getInfo(varargin{:}, 'fsdata');
0273     end
0274
0275     function out = SETS()
0276         out = {'Default'};
0277     end
0278
0279     function out = getDefaultPlist(set)
0280         switch set
0281             case 'Default'
0282                 out = plist();
0283             otherwise
0284                 error('### Unknown set [%s]', set);
0285         end
0286     end
0287
0288 end % End static methods
0289
0290 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0291 %                                Methods (static, private)                                %
0292 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0293
0294 methods (Static, Access = private)
0295     f = getFfromYFs(N,fs)
0296 end
0297
0298 end % End classdef
0299
```

applymethod

PURPOSE ^

APPLYMETHOD applies the given method to the input 2D data.

SYNOPSIS ^

function varargout = applymethod(varargin)

DESCRIPTION ^

```
APPLYMETHOD applies the given method to the input 2D data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: APPLYMETHOD applies the given method to the input 2D data.

CALL:      d = applymethod(d, pl)

INPUTS:    d      - a 2D data object (tsdata, fsdata, xydata)
           pl      - a plist of configuration options

PARAMETERS:

    'method' - the method to apply to the data
    'axis'   - which axis vector to apply the method to. Possible values
               are: 'X', 'Y', 'XY' [default: 'Y']
    'dim'    - the dimension of the chosen vector to apply the method
               to. This is necessary for functions like mean() when
               applied to matrices held in cdata objects. For tsdata,
               fsdata or xydata, this option has no effect.
               [default: 1]
    'option' - any additional option to pass to the method.

M-FILE INFO: Get information about this methods by calling
              >> fsdata.getInfo('applymethod')

              Get information about a specified set-plist by calling:
              >> fsdata.getInfo('applymethod', 'None')

VERSION:    $Id: applymethod.m,v 1.4 2008/09/04 15:29:30 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  function v = apply(v, method, dim, opt)
-  function ii = getInfo(varargin)
-  function pl = getDefaultPlist()

SOURCE CODE ^

```

0001 % APPLYMETHOD applies the given method to the input 2D data.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: APPLYMETHOD applies the given method to the input 2D data.
0005 %
0006 % CALL:          d = applymethod(d, pl)
0007 %
0008 % INPUTS:        d      - a 2D data object (tsdata, fsdata, xydata)
0009 %                pl      - a plist of configuration options
0010 %
0011 % PARAMETERS:
0012 %
0013 %         'method' - the method to apply to the data
0014 %         'axis'   - which axis vector to apply the method to. Possible values
0015 %                   are: 'X', 'Y', 'XY' [default: 'Y']
0016 %         'dim'    - the dimension of the chosen vector to apply the method
0017 %                   to. This is necessary for functions like mean() when
0018 %                   applied to matrices held in cdata objects. For tsdata,
0019 %                   fsdata or xydata, this option has no effect.
0020 %                   [default: 1]
0021 %         'option' - any additional option to pass to the method.
0022 %
0023 % M-FILE INFO: Get information about this methods by calling
0024 %               >> fsdata.getInfo('applymethod')
0025 %
0026 %               Get information about a specified set-plist by calling:
0027 %               >> fsdata.getInfo('applymethod', 'None')
0028 %
0029 % VERSION:      $Id: applymethod.m,v 1.4 2008/09/04 15:29:30 ingo Exp $
0030 %
0031 % HISTORY:      04-02-2007 M Hewitson
0032 %               Creation
0033 %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035
0036 function varargout = applymethod(varargin)
0037
0038 %%% Check if this is a call for parameters
0039 if utils.helper.isinfocall(varargin{:})
0040     varargout{1} = getInfo(varargin{3});
0041     return
0042 end
0043
0044 % Get data2D objects and plists
0045 ds = utils.helper.collect_objects(varargin(:), 'data2D');
0046 pl = utils.helper.collect_objects(varargin(:), 'plist');
0047
0048 % Combine with default plist
0049 pl = combine(pl, getDefaultPlist);
0050
0051 % Get the method to apply
0052 method = find(pl, 'method');
0053 % Get the axis we are dealing with
0054 axis = find(pl, 'axis');
0055 % Get the dimension to operate along
0056 dim = find(pl, 'dim');
0057 % Get any additional option
0058 opt = find(pl, 'option');
0059
0060 % Loop over data objects
0061 for jj=1: numel(ds)
0062     switch upper(axis)
0063     case 'X'
0064         ds(jj).x = apply(ds(jj).getX, method, dim, opt);
0065     case 'Y'
0066         ds(jj).y = apply(ds(jj).y, method, dim, opt);
0067     case 'XY'
0068         ds(jj).x = apply(ds(jj).getX, method, dim, opt);
0069         ds(jj).y = apply(ds(jj).y, method, dim, opt);
0070     otherwise
0071         error('### Unknown axis to operate on.');
```

%

```

0087 if ~isempty(dim) && ~isempty(opt)
0088     % User supplied a dimension and an option
0089     v = feval(method, v, dim, opt);
0090 elseif ~isempty(dim)
0091     % User supplied only a dimension
0092     v = feval(method, v, dim);
0093 elseif ~isempty(opt)
0094     % User supplied only an option
0095     v = feval(method, v, opt);
0096 else
0097     % User supplied only a method
0098     v = feval(method, v);
0099 end
0100 end
0101
0102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103 %
0104 % FUNCTION:      getInfo
0105 %
0106 % DESCRIPTION:  Get Info Object
0107 %
0108 % HISTORY:      11-07-07 M Hewitson
0109 %               Creation.
0110 %
0111 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0112
0113 function ii = getInfo(varargin)
0114     if nargin == 1 && strcmpi(varargin{1}, 'None')
0115         sets = {};
0116         pl = [];
0117     else
0118         sets = {'Default'};
0119         pl = getDefaultPlist;
0120     end
0121     % Build info object
0122     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id:
0123 applymethod.m,v 1.4 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0124 end
0125 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0126 %
0127 % FUNCTION:      getDefaultPlist
0128 %
0129 % DESCRIPTION:  Get Default Plist
0130 %
0131 % HISTORY:      11-07-07 M Hewitson
0132 %               Creation.
0133 %
0134 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0135
0136 function pl = getDefaultPlist()
0137     pl = plist('method', '', 'axis', 'y', 'dim', [], 'option', '');
0138 end
0139

```


applyoperator

PURPOSE ^

APPLYOPERATOR applys the given operator to the two input data objects.

SYNOPSIS ^

function varargout = applyoperator(varargin)

DESCRIPTION ^

```
APPLYOPERATOR applys the given operator to the two input data objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: APPLYOPERATOR applys the given operator to the two input
             2D data objects.

CALL:       d = applyoperator(d1, d2, pl)

INPUTS:     d1/2   - an ltpda_data object (cdata, tsdata, fsdata, xydata)
             pl    - a plist of configuration options

PARAMETERS: 'op'   - the operator to apply, e.g. 'plus'

M-FILE INFO: Get information about this methods by calling
              >> fsdata.getInfo('applyoperator')

              Get information about a specified set-plist by calling:
              >> fsdata.getInfo('applyoperator', 'set')

VERSION:    $Id: applyoperator.m,v 1.12 2008/09/04 15:29:30 ingo Exp $

HISTORY:    04-02-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function pl = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % APPLYOPERATOR applys the given operator to the two input data objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: APPLYOPERATOR applys the given operator to the two input
0005 %              2D data objects.
0006 %
0007 % CALL:       d = applyoperator(d1, d2, pl)
0008 %
0009 % INPUTS:     d1/2   - an ltpda_data object (cdata, tsdata, fsdata, xydata)
0010 %             pl    - a plist of configuration options
0011 %
```



```

0012 % PARAMETERS: 'op' - the operator to apply, e.g. 'plus'
0013 %
0014 % M-FILE INFO: Get information about this methods by calling
0015 % >> fsdata.getInfo('applyoperator')
0016 %
0017 % Get information about a specified set-plist by calling:
0018 % >> fsdata.getInfo('applyoperator', 'set')
0019 %
0020 % VERSION: $Id: applyoperator.m,v 1.12 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY: 04-02-2007 M Hewitson
0023 % Creation
0024 %
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0026
0027 function varargout = applyoperator(varargin)
0028
0029 %%% Check if this is a call for parameters
0030 if utils.helper.isinfocall(varargin{:})
0031     varargout{1} = getInfo(varargin{3});
0032     return
0033 end
0034
0035 % Get the objects we have in the correct order
0036 objs = [];
0037 for j=1:nargin
0038     if isa(varargin{j}, 'data2D') || isa(varargin{j}, 'cdata')
0039         objs = [objs {varargin{j}}];
0040     end
0041 end
0042
0043 % Get data2D objects and plists
0044 [pl] = utils.helper.collect_objects(varargin(:), 'plist');
0045
0046 % Combine with default plist
0047 pl = combine(pl, getDefaultPlist);
0048
0049 % Get the operator to apply
0050 op = find(pl, 'op');
0051
0052 if numel(objs) ~= 2
0053     error('### data2D/applyoperator requires two input data objects to work on.');
```

end

```

0055 %----- Add some rules here.
0056 % cdata
0057 % 1) time-base must match
0058 % 2) y dimensions must match or one must be a single value
0059 %
0060 %
0061 %%% Decide the type of the output object
0062 if isa(objs{1}, 'data2D')
0063     dout = objs{1};
0064     y1 = dout.y;
0065     y2 = objs{2}.y;
0066 elseif isa(objs{2}, 'data2D')
0067     dout = objs{2};
0068     y1 = objs{1}.y;
0069     y2 = dout.y;
0070 else
0071     dout = objs{1};
0072     y1 = dout.y;
0073     y2 = objs{2}.y;
0074 end
0075
0076 %%% Change for the following operations the shape of the axis
0077 %%% if the shape is different
0078 if strcmp(op, 'minus') || ...
0079     strcmp(op, 'plus') || ...
0080     strcmp(op, 'rdivide') || ...
0081     strcmp(op, 'times') || ...
0082     strcmp(op, 'power')
0083     s1 = size(y1);
0084     s2 = size(y2);
0085     if s1(1) == 1 && s2(1) ~= 1
0086         y2 = y2.';
0087     end
0088     if s1(2) == 1 && s2(2) ~= 1
0089         y2 = y2.';
0090     end
0091 end
0092
0093 dout.setY(feval(op, y1, y2));
0094
0095 varargout{1} = dout;
0096
0097 end

```

```
0098
0099 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100 %                               Local Functions                               %
0101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0102
0103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104 %
0105 % FUNCTION:      getInfo
0106 %
0107 % DESCRIPTION:  Get Info Object
0108 %
0109 % HISTORY:      11-07-07 M Hewitson
0110 %              Creation.
0111 %
0112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113
0114 function ii = getInfo(varargin)
0115     if nargin == 1 && strcmpi(varargin{1}, 'None')
0116         sets = {};
0117         pl   = [];
0118     else
0119         sets = {'Default'};
0120         pl   = getDefaultPlist;
0121     end
0122     % Build info object
0123     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id:
applyoperator.m,v 1.12 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0124 end
0125
0126 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0127 %
0128 % FUNCTION:      getDefaultPlist
0129 %
0130 % DESCRIPTION:  Get Default Plist
0131 %
0132 % HISTORY:      11-07-07 M Hewitson
0133 %              Creation.
0134 %
0135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0136
0137 function pl = getDefaultPlist()
0138     pl = plist('op', '');
0139 end
```

getX

PURPOSE ^

GETX Get the property 'x'.

SYNOPSIS ^

function varargout = getX(varargin)

DESCRIPTION ^

```
GETX Get the property 'x'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the property 'x'.

CALL:      val = obj.getX();
           val = obj.getX(idx);
           val = obj.getX(1:10);

INPUTS:    obj - must be a single data2D object.
           idx - index of the data samples

M-FILE INFO: Get information about this methods by calling
             >> fsdata.getInfo('getX')

             Get information about a specified set-plist by calling:
             >> fsdata.getInfo('getX', 'None')

VERSION:   $Id: getX.m,v 1.4 2008/09/04 15:29:30 ingo Exp $

HISTORY:   27-05-2008 Diepholz
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETX Get the property 'x'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the property 'x'.
0005 %
0006 % CALL:      val = obj.getX();
0007 %           val = obj.getX(idx);
0008 %           val = obj.getX(1:10);
0009 %
0010 % INPUTS:    obj - must be a single data2D object.
0011 %           idx - index of the data samples
0012 %
```

```

0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> fsdata.getInfo('getX')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> fsdata.getInfo('getX', 'None')
0018 %
0019 % VERSION: $Id: getX.m,v 1.4 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY: 27-05-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = getX(varargin)
0027
0028 %% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 %% get idx
0035 idx = [];
0036 if nargin == 2
0037     if isnumeric(varargin{2})
0038         idx = varargin{2};
0039     else
0040         error('### Unknown arguments to getX.');
```

```
0098 %
0099 % HISTORY:      11-07-07 M Hewitson
0100 %              Creation.
0101 %
0102 %%%%%%%%%%%%%%
0103
0104 function plo = getDefaultPlist()
0105     plo = plist();
0106 end
0107
```

getY

PURPOSE ^

GETY Get the property 'y'.

SYNOPSIS ^

function varargout = getY(varargin)

DESCRIPTION ^

```
GETY Get the property 'y'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the property 'y'.

CALL:      val = obj.getY();
           val = obj.getY(idx);
           val = obj.getY(1:10);

INPUTS:    obj - must be a single data2D object.
           idx - index of the data samples

M-FILE INFO: Get information about this methods by calling
           >> fsdata.getInfo('getY')

           Get information about a specified set-plist by calling:
           >> fsdata.getInfo('getY', 'None')

VERSION:   $Id: getY.m,v 1.3 2008/09/04 15:29:30 ingo Exp $

HISTORY:   27-05-2008 Diepholz
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETY Get the property 'y'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the property 'y'.
0005 %
0006 % CALL:      val = obj.getY();
0007 %           val = obj.getY(idx);
0008 %           val = obj.getY(1:10);
0009 %
0010 % INPUTS:    obj - must be a single data2D object.
0011 %           idx - index of the data samples
0012 %
```

```

0013 % M-FILE INFO: Get information about this methods by calling
0014 %      >> fsdata.getInfo('getY')
0015 %
0016 %      Get information about a specified set-plist by calling:
0017 %      >> fsdata.getInfo('getY', 'None')
0018 %
0019 % VERSION:      $Id: getY.m,v 1.3 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY:      27-05-2008 Diepholz
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = getY(varargin)
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 %%% get idx
0035 idx = [];
0036 if nargin == 2
0037     if isnumeric(varargin{2})
0038         idx = varargin{2};
0039     else
0040         error('### Unknown arguments to getY.');
```

```
0098   plo = plist();  
0099   end  
0100
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setX

PURPOSE ^

SETX Set the property 'x'.

SYNOPSIS ^

function varargout = setX(varargin)

DESCRIPTION ^

```
SETX Set the property 'x'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'x'.

CALL:
    obj.setX([1 2 3]);
    obj = obj.setX([1 2 3]); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setX')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setX', 'None')

VERSION:
    $Id: setX.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETX Set the property 'x'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'x'.
0005 %
0006 % CALL:
0007 %     obj.setX([1 2 3]);
0008 %     obj = obj.setX([1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setX')
0015 %
0016 %     Get information about a specified set-plist by calling:
```

```
0015 %                >> fsdata.getInfo('setX', 'None')
0016 %
0017 % VERSION:      $Id: setX.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = setX(varargin)
0025
0026     %% Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     obj = varargin{1};
0033     val = varargin{2};
0034
0035     %% decide whether we modify the pz-object, or create a new one.
0036     obj = copy(obj, nargout);
0037
0038     %% set 'x'
0039     obj.x = val;
0040
0041     varargout{1} = obj;
0042 end
0043
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                                Local Functions                                %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id: setX.m,v
0069 1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0070 end
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081
0082 function plo = getDefaultPlist()
0083     plo = plist('x', '');
0084 end
0085
```

setXY

PURPOSE ^

SETXY Set the property 'xy'.

SYNOPSIS ^

function varargout = setXY(varargin)

DESCRIPTION ^

```
SETXY Set the property 'xy'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'xy'.

CALL:
    obj.setXY([1 2 3], [1 2 3]);
    obj = obj.setXY([1 2 3], [1 2 3]); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setXY')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setXY', 'None')

VERSION:
    $Id: setXY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETXY Set the property 'xy'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'xy'.
0005 %
0006 % CALL:
0007 %     obj.setXY([1 2 3], [1 2 3]);
0008 %     obj = obj.setXY([1 2 3], [1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setXY')
0015 %
0016 %     Get information about a specified set-plist by calling:
```

```
0015 %                >> fsdata.getInfo('setXY', 'None')
0016 %
0017 % VERSION:      $Id: setXY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = setXY(varargin)
0025
0026     %% Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     obj = varargin{1};
0033     x   = varargin{2};
0034     y   = varargin{3};
0035
0036     %% decide whether we modify the pz-object, or create a new one.
0037     obj = copy(obj, nargout);
0038
0039     %% set 'x' and 'y'
0040     obj.x = x;
0041     obj.y = y;
0042
0043     varargout{1} = obj;
0044 end
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %                                Local Functions                                %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 % FUNCTION:      getInfo
0053 %
0054 % DESCRIPTION:   Get Info Object
0055 %
0056 % HISTORY:      11-07-07 M Hewitson
0057 %                Creation.
0058 %
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl   = [];
0065     else
0066         sets = {'Default'};
0067         pl   = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id: setXY.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0071 end
0072
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getDefaultPlist
0076 %
0077 % DESCRIPTION:   Get Default Plist
0078 %
0079 % HISTORY:      11-07-07 M Hewitson
0080 %                Creation.
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 function plo = getDefaultPlist()
0085     plo = plist('xy', '');
0086 end
0087
```

setXunits

PURPOSE ^

SETXUNITS Set the property 'xunits'.

SYNOPSIS ^

function varargout = setXunits(varargin)

DESCRIPTION ^

```
SETXUNITS Set the property 'xunits'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'xunits'.

CALL:
    obj.setXunits('units');
    obj = obj.setXunits('units'); create copy of the object

INPUTS:
    obj - must be a single data2D object.
    units - unit object or a valid string which can be transform
            into a unit object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setXunits')

            Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setXunits', 'None')

VERSION:
    $Id: setXunits.m,v 1.6 2008/09/05 14:16:30 hewitson Exp $

HISTORY:
    27-05-2008 Diepholz
                Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETXUNITS Set the property 'xunits'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'xunits'.
0005 %
0006 % CALL:
0007 %     obj.setXunits('units');
0008 %     obj = obj.setXunits('units'); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %     units - unit object or a valid string which can be transform
               into a unit object.
```

```

0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> fsdata.getInfo('setXunits')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> fsdata.getInfo('setXunits', 'None')
0018 %
0019 % VERSION: $Id: setXunits.m,v 1.6 2008/09/05 14:16:30 hewitson Exp $
0020 %
0021 % HISTORY: 27-05-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = setXunits(varargin)
0027
0028 %% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 obj = varargin{1};
0035 val = varargin{2};
0036
0037 %% decide whether we modify the pz-object, or create a new one.
0038 obj = copy(obj, nargout);
0039
0040 %% set 'xunits'
0041 if ischar(val)
0042     obj.xunits = unit(val);
0043 elseif iscell(val)
0044     obj.xunits = val{1};
0045 else
0046     obj.xunits = val;
0047 end
0048
0049 varargout{1} = obj;
0050 end
0051
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053 % Local Functions %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057 %
0058 % FUNCTION: getInfo
0059 %
0060 % DESCRIPTION: Get Info Object
0061 %
0062 % HISTORY: 11-07-07 M Hewitson
0063 % Creation.
0064 %
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066
0067 function ii = getInfo(varargin)
0068     if nargin == 1 && strcmpi(varargin{1}, 'None')
0069         sets = {};
0070         pl = [];
0071     else
0072         sets = {'Default'};
0073         pl = getDefaultPlist;
0074     end
0075     % Build info object
0076     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id:
setXunits.m,v 1.6 2008/09/05 14:16:30 hewitson Exp $', sets, pl);
0077 end
0078
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080 %
0081 % FUNCTION: getDefaultPlist
0082 %
0083 % DESCRIPTION: Get Default Plist
0084 %
0085 % HISTORY: 11-07-07 M Hewitson
0086 % Creation.
0087 %
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089
0090 function plo = getDefaultPlist()
0091     plo = plist('xunits', unit);
0092 end
0093

```

setY

PURPOSE ^

SETY Set the property 'y'.

SYNOPSIS ^

function varargout = setY(varargin)

DESCRIPTION ^

```
SETY Set the property 'y'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'y'.

CALL:
    obj.setY([1 2 3]);
    obj = obj.setY([1 2 3]); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setY')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setY', 'None')

VERSION:
    $Id: setY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETY Set the property 'y'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'y'.
0005 %
0006 % CALL:
0007 %     obj.setY([1 2 3]);
0008 %     obj = obj.setY([1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setY')
0015 %
0016 %     Get information about a specified set-plist by calling:
```

```
0015 %                >> fsdata.getInfo('setY', 'None')
0016 %
0017 % VERSION:      $Id: setY.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = setY(varargin)
0025
0026     %% Check if this is a call for parameters
0027     if utils.helper.isinfocall(varargin{:})
0028         varargout{1} = getInfo(varargin{3});
0029         return
0030     end
0031
0032     obj = varargin{1};
0033     val = varargin{2};
0034
0035     %% decide whether we modify the pz-object, or create a new one.
0036     obj = copy(obj, nargout);
0037
0038     %% set 'y'
0039     obj.y = val;
0040
0041     varargout{1} = obj;
0042 end
0043
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                                Local Functions                                %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id: setY.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0069 end
0070
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081
0082 function plo = getDefaultPlist()
0083     plo = plist('y', '');
0084 end
0085
```


setYunits

PURPOSE ^

SEYUNITS Set the property 'yunits'.

SYNOPSIS ^

function varargout = setYunits(varargin)

DESCRIPTION ^

```
SEYUNITS Set the property 'yunits'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'yunits'.

CALL:
    obj.setYunits('units');
    obj = obj.setYunits('units'); create copy of the object

INPUTS:
    obj - must be a single data2D object.
    units - unit object or a valid string which can be transform
           into a unit object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setYunits')

           Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setYunits', 'None')

VERSION:
    $Id: setYunits.m,v 1.6 2008/09/05 14:16:30 hewitson Exp $

HISTORY:
    27-05-2008 Diepholz
                Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

 [function ii = getInfo\(varargin\)](#)

 [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SEYUNITS Set the property 'yunits'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'yunits'.
0005 %
0006 % CALL:
0007 %     obj.setYunits('units');
0008 %     obj = obj.setYunits('units'); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %     units - unit object or a valid string which can be transform
              into a unit object.
```

```

0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> fsdata.getInfo('setYunits')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> fsdata.getInfo('setYunits', 'None')
0018 %
0019 % VERSION: $Id: setYunits.m,v 1.6 2008/09/05 14:16:30 hewitson Exp $
0020 %
0021 % HISTORY: 27-05-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = setYunits(varargin)
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 obj = varargin{1};
0035 val = varargin{2};
0036
0037 %%% decide whether we modify the pz-object, or create a new one.
0038 obj = copy(obj, nargout);
0039
0040 %%% set 'yunits'
0041 if ischar(val)
0042     obj.yunits = unit(val);
0043 elseif iscell(val)
0044     obj.yunits = val{1};
0045 else
0046     obj.yunits = val;
0047 end
0048
0049 varargout{1} = obj;
0050 end
0051
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053 % Local Functions %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057 %
0058 % FUNCTION: getInfo
0059 %
0060 % DESCRIPTION: Get Info Object
0061 %
0062 % HISTORY: 11-07-07 M Hewitson
0063 % Creation.
0064 %
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066
0067 function ii = getInfo(varargin)
0068     if nargin == 1 && strcmpi(varargin{1}, 'None')
0069         sets = {};
0070         pl = [];
0071     else
0072         sets = {'Default'};
0073         pl = getDefaultPlist;
0074     end
0075     % Build info object
0076     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id:
setYunits.m,v 1.6 2008/09/05 14:16:30 hewitson Exp $', sets, pl);
0077 end
0078
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080 %
0081 % FUNCTION: getDefaultPlist
0082 %
0083 % DESCRIPTION: Get Default Plist
0084 %
0085 % HISTORY: 11-07-07 M Hewitson
0086 % Creation.
0087 %
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089
0090 function plo = getDefaultPlist()
0091     plo = plist('yunits', unit);
0092 end
0093

```

copy

PURPOSE ^

COPY Make copy of fsdata objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of fsdata objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of fsdata objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> fsdata.getInfo('copy')

VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of fsdata objects depending of the second input
-  [fsdata](#) FSDATA frequency-series object class constructor.

This function is called by:

-  [copy](#) COPY Make copy of fsdata objects depending of the second input
-  [fsdata](#) FSDATA frequency-series object class constructor.
-  [setEnbw](#) SETENBW Set the property 'enbw'.
-  [setFs](#) SETFS Set the property 'fs'.
-  [setNavs](#) SETNAVS Set the property 'navs'.
-  [setT0](#) SETT0 Set the property 't0'.

SOURCE CODE ^

```
0001 % COPY Make copy of fsdata objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of fsdata objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> fsdata.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
```

```
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020 if deepcopy
0021     ps = properties('fsdata');
0022     s = size(old);
0023     obj(s(1),s(2)) = fsdata;
0024     for kk=1:numel(old)
0025         utils.helper.msg(utils.const.msg.OPROC2, 'copying fsdata');
0026         for jj=1:length(ps)
0027             obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028         end
0029         % copy time object
0030         obj(kk).t0 = copy(old(kk).t0, 1);
0031         obj(kk).version = fsdata.VEROUT;
0032     end
0033 else
0034     obj = old;
0035 end
0036 varargout{1} = obj;
0037 end
0038
```

setEnbw

PURPOSE ^

SETENBW Set the property 'enbw'.

SYNOPSIS ^

function varargout = setEnbw(varargin)

DESCRIPTION ^

```
SETENBW Set the property 'enbw'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'enbw'.

CALL:
    obj.setEnbw(val);
    obj = obj.setEnbw(val); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setEnbw')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setEnbw', 'None')

VERSION:
    $Id: setEnbw.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of fsdata objects depending of the second input

This function is called by:

 [fsdata](#) FSDATA frequency-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETENBW Set the property 'enbw'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'enbw'.
0005 %
0006 % CALL:
0007 %     obj.setEnbw(val);
0008 %     obj = obj.setEnbw(val); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setEnbw')
0015 %
0016 %     Get information about a specified set-plist by calling:
0017 %     >> fsdata.getInfo('setEnbw', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setEnbw.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %           >> fsdata.getInfo('setEnbw')
0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> fsdata.getInfo('setEnbw', 'None')
0016 %
0017 % VERSION:      $Id: setEnbw.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setEnbw(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'enbw'
0039 obj.enbw = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                               Local Functions                               %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'fsdata', '', utils.const.categories.internal, '$Id: setEnbw.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('enbw', '');
0084 end
0085
```

setFs

PURPOSE ^

SETFS Set the property 'fs'.

SYNOPSIS ^

function varargout = setFs(varargin)

DESCRIPTION ^

```
SETFS Set the property 'fs'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'fs'.

CALL:
    obj.setFs(12);
    obj = obj.setFs(12); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setFs')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setFs', 'None')

VERSION:
    $Id: setFs.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of fsdata objects depending of the second input

This function is called by:

 [fsdata](#) FSDATA frequency-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETFS Set the property 'fs'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'fs'.
0005 %
0006 % CALL:
0007 %     obj.setFs(12);
0008 %     obj = obj.setFs(12); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setFs')
0015 %
0016 %     Get information about a specified set-plist by calling:
0017 %     >> fsdata.getInfo('setFs', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setFs.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %           >> fsdata.getInfo('setFs')
0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> fsdata.getInfo('setFs', 'None')
0016 %
0017 % VERSION:      $Id: setFs.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setFs(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'fs'
0039 obj.fs = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                               Local Functions                               %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl   = [];
0063     else
0064         sets = {'Default'};
0065         pl   = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'fsdata', '', utils.const.categories.internal, '$Id: setFs.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('fs', '');
0084 end
0085
```


setNavs

PURPOSE ^

SETNAVS Set the property 'navs'.

SYNOPSIS ^

function varargout = setNavs(varargin)

DESCRIPTION ^

```
SETNAVS Set the property 'navs'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'navs'.

CALL:
    obj.setNavs(val);
    obj = obj.setNavs(val); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setNavs')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setNavs', 'None')

VERSION:
    $Id: setNavs.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of fsdata objects depending of the second input

This function is called by:

 [fsdata](#) FSDATA frequency-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETNAVS Set the property 'navs'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'navs'.
0005 %
0006 % CALL:
0007 %     obj.setNavs(val);
0008 %     obj = obj.setNavs(val); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setNavs')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 %     >> fsdata.getInfo('setNavs', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setNavs.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %           >> fsdata.getInfo('setNavs')
0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> fsdata.getInfo('setNavs', 'None')
0016 %
0017 % VERSION:      $Id: setNavs.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setNavs(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'navs'
0039 obj.navs = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                               Local Functions                               %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'fsdata', '', utils.const.categories.internal, '$Id: setNavs.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('navs', '');
0084 end
0085
```

setT0

PURPOSE ^

SETT0 Set the property 't0'.

SYNOPSIS ^

function varargout = setT0(varargin)

DESCRIPTION ^

```
SETT0 Set the property 't0'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 't0'.

CALL:
    obj.setT0(time(0));
    obj = obj.setT0(time(0)); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setT0')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setT0', 'None')

VERSION:
    $Id: setT0.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of fsdata objects depending of the second input

This function is called by:

 [fsdata](#) FSDATA frequency-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETT0 Set the property 't0'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 't0'.
0005 %
0006 % CALL:
0007 %     obj.setT0(time(0));
0008 %     obj = obj.setT0(time(0)); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> fsdata.getInfo('setT0')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 %     >> fsdata.getInfo('setT0', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setT0.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %
0026 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %                >> fsdata.getInfo('setT0')
0013 %
0014 %                Get information about a specified set-plist by calling:
0015 %                >> fsdata.getInfo('setT0', 'None')
0016 %
0017 % VERSION:      $Id: setT0.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setT0(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 't0'
0039 obj.t0 = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                Local Functions
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'fsdata', '', utils.const.categories.internal, '$Id: setT0.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('t0', '');
0084 end
0085
```

char

PURPOSE ^

CHAR convert a ltpda_data-object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
CHAR convert a ltpda_data-object into a string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: CHAR convert a ltpda_data object into a string.

CALL:      string = char(fsdata)

M-FILE INFO: Get information about this methods by calling
              >> fsdata.getInfo('char')

              Get information about a specified set-plist by calling:
              >> fsdata.getInfo('char', 'set')

VERSION:    $Id: char.m,v 1.4 2008/09/04 15:29:30 ingo Exp $

HISTORY:    23-07-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a ltpda_data-object into a string.

This function is called by:

-  [char](#) CHAR convert a ltpda_data-object into a string.
-  [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a ltpda_data-object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a ltpda_data object into a string.
0005 %
0006 % CALL:      string = char(fsdata)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> fsdata.getInfo('char')
0010 %
0011 %              Get information about a specified set-plist by calling:
0012 %              >> fsdata.getInfo('char', 'set')
```

```

0013 %
0014 % VERSION:      $Id: char.m,v 1.4 2008/09/04 15:29:30 ingo Exp $
0015 %
0016 % HISTORY:      23-07-2007 Diepholz
0017 %                Creation
0018 %
0019 %%%%%%%%%%%%%%
0020 %
0021 function varargout = char(varargin)
0022
0023 %%% Check if this is a call for parameters
0024 if utils.helper.isinfocall(varargin{:})
0025     varargout{1} = getInfo(varargin{3});
0026     return
0027 end
0028
0029 % Collect all ltpda_data-objects
0030 objs = utils.helper.collect_objects(varargin(:), 'data2D');
0031
0032 %%% Add the size of the data (all)
0033 pstr = sprintf('Ndata=[%sx%s]', num2str(size(objs.y,1)), num2str(size(objs.y,2)));
0034
0035 %%% Add the sample rate of data (tsdata, fsdata)
0036 if isprop(objs, 'fs')
0037     pstr = sprintf('%s fs=%g,', pstr, objs.fs);
0038 end
0039
0040 %%% Add the length of this time-series in seconds (tsdata)
0041 if isprop(objs, 'nsecs')
0042     pstr = sprintf('%s nsecs=%d,', pstr, objs.nsecs);
0043 end
0044
0045 %%% Add number of averages (fsdata)
0046 if isprop(objs, 'navs')
0047     pstr = sprintf('%s navs=%d,', pstr, objs.navs);
0048 end
0049
0050 %%% Add time-stamp of the first data sample (fsdata, tsdata)
0051 if isprop(objs, 't0')
0052     pstr = sprintf('%s t0=%s,', pstr, char(objs.t0));
0053 end
0054
0055 pstr = pstr(1:end-1);
0056
0057 varargout{1} = pstr;
0058 end
0059
0060 %%%%%%%%%%%%%%
0061 % Local Functions
0062 %%%%%%%%%%%%%%
0063 %
0064 %%%%%%%%%%%%%%
0065 %
0066 % FUNCTION:      getInfo
0067 %
0068 % DESCRIPTION:   Get Info Object
0069 %
0070 % HISTORY:      11-07-07 M Hewitson
0071 %                Creation.
0072 %
0073 %%%%%%%%%%%%%%
0074
0075 function ii = getInfo(varargin)
0076     if nargin == 1 && strcmpi(varargin{1}, 'None')
0077         sets = {};
0078         pl    = [];
0079     else
0080         sets = {'Default'};
0081         pl    = getDefaultPlist;
0082     end
0083     % Build info object
0084     ii = minfo(mfilename, 'data2D', '', utils.const.categories.output, '$Id: char.m,v 1.4
2008/09/04 15:29:30 ingo Exp $', sets, pl);
0085 end
0086
0087 %%%%%%%%%%%%%%
0088 %
0089 % FUNCTION:      getDefaultPlist
0090 %
0091 % DESCRIPTION:   Get Default Plist
0092 %
0093 % HISTORY:      11-07-07 M Hewitson
0094 %                Creation.
0095 %
0096 %%%%%%%%%%%%%%
0097

```

```
0098 function plo = getDefaultPlist()
0099     plo = plist();
0100 end
0101
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

display

PURPOSE ^

DISPLAY implement terminal display for fsdata object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY implement terminal display for fsdata object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY implement terminal display for fsdata object.

M-FILE INFO: Get information about this methods by calling
>> fsdata.getInfo('display')

Get information about a specified set-plist by calling:
>> fsdata.getInfo('display', 'None')

VERSION:      $Id: display.m,v 1.14 2008/09/04 15:29:30 ingo Exp $

HISTORY:      31-01-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [fsdata](#) FSDATA frequency-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY implement terminal display for fsdata object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY implement terminal display for fsdata object.
0005 %
0006 % M-FILE INFO: Get information about this methods by calling
0007 % >> fsdata.getInfo('display')
0008 %
0009 % Get information about a specified set-plist by calling:
0010 % >> fsdata.getInfo('display', 'None')
0011 %
0012 % VERSION:      $Id: display.m,v 1.14 2008/09/04 15:29:30 ingo Exp $
0013 %
0014 % HISTORY:      31-01-2007 M Hewitson
0015 %               Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 function varargout = display(varargin)
```



```

0020
0021 %%% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 fsdatas = utils.helper.collect_objects(varargin(:), 'fsdata');
0028
0029 txt = {};
0030
0031 for i=1:numel(fsdatas)
0032     fsd = fsdatas(i);
0033
0034     banner_start = sprintf('----- fsdata %02d -----', i);
0035     txt{end+1} = banner_start;
0036     txt{end+1} = ' ';
0037
0038     xdata = fsdatas(i).x;
0039     ydata = fsdatas(i).y;
0040     xinfo = size(xdata);
0041     yinfo = size(ydata);
0042
0043     txt{end+1} = sprintf('      fs:  %g', fsdatas(i).fs);
0044     txt{end+1} = sprintf('      x:  [%d %d], %s', xinfo(1), xinfo(2), class(xdata));
0045     txt{end+1} = sprintf('      y:  [%d %d], %s', yinfo(1), yinfo(2), class(ydata));
0046     txt{end+1} = sprintf('xunits: %s', char(fsdatas(i).xunits));
0047     txt{end+1} = sprintf('yunits: %s', char(fsdatas(i).yunits));
0048     txt{end+1} = sprintf('t0:    %s', char(fsdatas(i).t0));
0049
0050     banner_end(1:length(banner_start)) = '-';
0051     txt{end+1} = banner_end;
0052
0053     txt{end+1} = ' ';
0054 end
0055
0056 %%% Prepare output
0057 if nargout == 0
0058     for ii=1:length(txt)
0059         disp(txt{ii});
0060     end
0061 elseif nargout == 1
0062     varargout{1} = txt;
0063 end
0064
0065 end
0066
0067 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0068 %                                     Local Functions                                     %
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0070
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0072 %
0073 % FUNCTION:      getInfo
0074 %
0075 % DESCRIPTION:  Get Info Object
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0081
0082 function ii = getInfo(varargin)
0083     if nargin == 1 && strcmpi(varargin{1}, 'None')
0084         sets = {};
0085         pl = [];
0086     else
0087         sets = {'Default'};
0088         pl = getDefaultPlist;
0089     end
0090     % Build info object
0091     ii = minfo(mfilename, 'fsdata', '', utils.const.categories.output, '$Id: display.m,v
1.14 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0092 end
0093
0094 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0095 %
0096 % FUNCTION:      getDefaultPlist
0097 %
0098 % DESCRIPTION:  Get Default Plist
0099 %
0100 % HISTORY:      11-07-07 M Hewitson
0101 %                Creation.
0102 %
0103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0104

```

```
0105 function plo = getDefaultPlist()
0106     plo = plist();
0107 end
0108
```

tsdata

PURPOSE ^

TSDATA time-series object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
TSDATA time-series object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: TSDATA time-series object class constructor.
             Create a time-series data object. If the time-series object
             is determined to be evenly sampled, no x vector is stored.

SUPER CLASSES: data2D < ltpda_data < ltpda_nuo < ltpda_obj

PROPERTIES:

  Inherit Properties (read only)
    version - cvs-version string
    xunits   - units of the y-axis
    yunits   - units of the y-axis
    x        - data values of the x-axis
    y        - data values of the y-axis

  Properties (read only)
    fs       - sample rate of data
    nsecs    - the length of this time-series in seconds
    t0       - time-stamp of the first data sample in UTC format yyyy-mm-dd HH:MM:SS

CONSTRUCTORS:

  ts = tsdata()           - creates a blank time-series object
  ts = tsdata(y)          - creates a time-series object with the given
                           y-data.
  ts = tsdata(x,y)        - creates a time-series object with the given
                           (x,y)-data. The sample rate is then set using
                           the private method fitfs(). This computes the
                           best sample rate that fits the data. If the
                           data is evenly sampled, the sample rate is set
                           as 1/median(diff(x)) and the x data is then
                           not stored (empty vector).
  ts = tsdata(y,fs)       - creates a time-series object with the given
                           y-data. The data is assumed to be evenly sampled
                           at the given sample rate with the first sample
                           assigned time 0. No x vector is created.
  ts = tsdata(y,t0)       - creates a time-series object with the given
                           y-data. The data are assumed to be evenly
                           sampled at 1Hz. The first sample is assumed to
                           be at 0s offset from t0 and t0 is set to the
                           user specified value.
  ts = tsdata(x,y,fs)     - creates a time-series object with the given
                           x/y data vectors. The sample rate is set to
                           fs.
  ts = tsdata(x,y,t0)     - creates a time-series object with the given
                           x/y data vectors. The t0 property is set to
                           the supplied t0 and the sample rate is
                           computed from the x vector using fitfs(). If
                           the data is found to be evenly sampled, the
                           x vector is discarded.
  ts = tsdata(y,fs,t0)    - creates a time-series object with the given
                           y-data. The data are assumed to be evenly
                           sampled at fs and the first sample is
                           assumed to be taken at time t0. No x vector
                           is generated.
  ts = tsdata(x,y,fs,t0) - creates a time-series object with the given
                           x-data, y-data, fs and t0.

TSDATA METHODS:
```

```
Defined Abstract methods:
char          - returns one character string which represents the object
copy          - copies an object
display       - displays an object
update_struct - updates a object structure to the current tbx-version

Public methods:
setT0         - set the property 't0'
setFs         - set the property 'fs'
setNsecs      - set the property 'nsecs'
getX         - overload the method in data2D
growT         - grows the time (x) vector if it is empty
fixNsecs      - fixes the number of seconds
collapseX     - checks whether the x vector is evenly sampled and
                  then remove it

Static methods:
fitfs         - computes the sample rate of the input data.

M-FILE INFO:  The following call returns an minfo object that contains
                information about the tsdata constructor:
                >> info = tsdata.getInfo
or             >> info = tsdata.getInfo('tsdata')

                You can get information about class methods by calling:
                >> info = tsdata.getInfo(method)
e.g. >> info = tsdata.getInfo('eq')

                You can also restrict the sets of parameters contained in
                the minfo object by calling:
                >> info = tsdata.getInfo(method, set)
e.g. >> info = tsdata.getInfo('tsdata', 'Default')

VERSION:  $Id: tsdata.m,v 1.57 2008/09/03 16:39:37 hewitson Exp $

HISTORY:  30-01-2007 Hewitson
            Creation

SEE ALSO:  tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [collapseX](#) COLLAPSEX Checks whether the x vector is evenly sampled and then removes it
-  [copy](#) COPY Make copy of tsdata objects depending of the second input
-  [display](#) DISPLAY overloads display functionality for tsdata objects.
-  [fitfs](#) FITFS computes the sample rate of the input data. If the min and max sample
-  [fixNsecs](#) FIXNSECS fixes the numer of seconds.
-  [getX](#) GETX Get the property 'x'.
-  [growT](#) GROWT grows the time (x) vector if it is empty.
-  [setFs](#) SETFS Set the property 'fs'.
-  [setNsecs](#) SETNSECS Set the property 'nsecs'.
-  [setT0](#) SETT0 Set the property 't0'.
-  [tsdata](#) TSDATA time-series object class constructor.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

This function is called by:

-  [copy](#) COPY Make copy of tsdata objects depending of the second input
-  [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function obj = set.t0\(obj, val\)](#)
- 

```

function obj = set.fs(obj, val)
function obj = set.nsecs(obj, val)
function obj = tsdata(varargin)
function out = VEROUT()
function ii = getInfo(varargin)
function out = SETS()
function out = getDefaultPlist(set)

```

SOURCE CODE ^

```

0001 % TSDATA time-series object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: TSDATA time-series object class constructor.
0005 %               Create a time-series data object. If the time-series object
0006 %               is determined to be evenly sampled, no x vector is stored.
0007 %
0008 % SUPER CLASSES: data2D < ltpda_data < ltpda_nuo < ltpda_obj
0009 %
0010 % PROPERTIES:
0011 %
0012 %     Inherit Properties (read only)
0013 %         version - cvs-version string
0014 %         xunits  - units of the y-axis
0015 %         yunits  - units of the y-axis
0016 %         x       - data values of the x-axis
0017 %         y       - data values of the y-axis
0018 %
0019 %     Properties (read only)
0020 %         fs      - sample rate of data
0021 %         nsecs   - the length of this time-series in seconds
0022 %         t0      - time-stamp of the first data sample in UTC format yyyy-mm-dd HH:MM:SS
0023 %
0024 % CONSTRUCTORS:
0025 %
0026 %         ts = tsdata()           - creates a blank time-series object
0027 %         ts = tsdata(y)         - creates a time-series object with the given
0028 %                                y-data.
0029 %         ts = tsdata(x,y)       - creates a time-series object with the given
0030 %                                (x,y)-data. The sample rate is then set using
0031 %                                the private method fitfs(). This computes the
0032 %                                best sample rate that fits the data. If the
0033 %                                data is evenly sampled, the sample rate is set
0034 %                                as 1/median(diff(x)) and the x data is then
0035 %                                not stored (empty vector).
0036 %         ts = tsdata(y,fs)      - creates a time-series object with the given
0037 %                                y-data. The data is assumed to be evenly sampled
0038 %                                at the given sample rate with the first sample
0039 %                                assigned time 0. No x vector is created.
0040 %         ts = tsdata(y,t0)      - creates a time-series object with the given
0041 %                                y-data. The data are assumed to be evenly
0042 %                                sampled at 1Hz. The first sample is assumed to
0043 %                                be at 0s offset from t0 and t0 is set to the
0044 %                                user specified value.
0045 %         ts = tsdata(x,y,fs)    - creates a time-series object with the given
0046 %                                x/y data vectors. The sample rate is set to
0047 %                                fs.
0048 %         ts = tsdata(x,y,t0)    - creates a time-series object with the given
0049 %                                x/y data vectors. The t0 property is set to
0050 %                                the supplied t0 and the sample rate is
0051 %                                computed from the x vector using fitfs(). If
0052 %                                the data is found to be evenly sampled, the
0053 %                                x vector is discarded.
0054 %         ts = tsdata(y,fs,t0)   - creates a time-series object with the given
0055 %                                y-data. The data are assumed to be evenly
0056 %                                sampled at fs and the first sample is
0057 %                                assumed to be taken at time t0. No x vector
0058 %                                is generated.
0059 %         ts = tsdata(x,y,fs,t0) - creates a time-series object with the given
0060 %                                x-data, y-data, fs and t0.
0061 %
0062 % TSDATA METHODS:
0063 %
0064 %     Defined Abstract methods:
0065 %         char          - returns one character string which represents the object
0066 %         copy          - copies an object
0067 %         display        - displays an object
0068 %         update_struct - updates a object structure to the current tbx-version
0069 %

```

```

0070 %      Public methods:
0071 %          setT0      - set the property 't0'
0072 %          setFs      - set the property 'fs'
0073 %          setNsecs   - set the property 'nsecs'
0074 %          getX       - overload the method in data2D
0075 %          growT      - grows the time (x) vector if it is empty
0076 %          fixNsecs   - fixes the number of seconds
0077 %          collapseX  - checks whether the x vector is evenly sampled and
0078 %                      then remove it
0079 %
0080 %      Static methods:
0081 %          fitfs      - computes the sample rate of the input data.
0082 %
0083 %
0084 % M-FILE INFO: The following call returns an minfo object that contains
0085 %               information about the tsdata constructor:
0086 %               >> info = tsdata.getInfo
0087 %               or   >> info = tsdata.getInfo('tsdata')
0088 %
0089 %               You can get information about class methods by calling:
0090 %               >> info = tsdata.getInfo(method)
0091 %               e.g. >> info = tsdata.getInfo('eq')
0092 %
0093 %               You can also restrict the sets of parameters contained in
0094 %               the minfo object by calling:
0095 %               >> info = tsdata.getInfo(method, set)
0096 %               e.g. >> info = tsdata.getInfo('tsdata', 'Default')
0097 %
0098 % VERSION:  $Id: tsdata.m,v 1.57 2008/09/03 16:39:37 hewitson Exp $
0099 %
0100 % HISTORY:  30-01-2007 Hewitson
0101 %            Creation
0102 %
0103 % SEE ALSO: tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata
0104 %
0105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106 %
0107 classdef tsdata < data2D
0108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109 %
0110 %      Property definition
0111 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0112 %
0113 %----- Public (read/write) Properties -----
0114 properties
0115 end
0116 %
0117 %----- Protected read-only Properties -----
0118 properties (SetAccess = protected)
0119     t0      = time(0);
0120     fs      = NaN;
0121     nsecs   = 0;
0122     version = '$Id: tsdata.m,v 1.57 2008/09/03 16:39:37 hewitson Exp $';
0123 end
0124 %
0125 %----- Private Properties -----
0126 properties (GetAccess = protected, SetAccess = protected)
0127 end
0128 %
0129 %----- Abstract Properties -----
0130 properties (Abstract = true, SetAccess = protected)
0131 end
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133 %
0134 %      Check property setting
0135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0136 %
0137 methods
0138     function obj = set.t0(obj, val)
0139         if (~isa(val, 'time') && ~ischar(val) && ~isnumeric(val)) || isempty(val)
0140             error('### The value for the property ''t0'' must be a string, a number or a
time object');
0141         end
0142         if ischar(val) || isnumeric(val)
0143             obj.t0 = time(val);
0144         else
0145             obj.t0 = val;
0146         end
0147     end
0148     function obj = set.fs(obj, val)
0149         if ~isnumeric(val) || isempty(val) || ~isreal(val) || val < 0
0150             error('### The value for the property ''fs'' must be a real positive number');
0151         end
0152         obj.fs = val;
0153     end
0154     function obj = set.nsecs(obj, val)

```

```

0155     if ~isnumeric(val) || isempty(val) || ~isreal(val) || val < 0
0156         error('### The value for the property ''nsecs'' must be a real positive
number');
0157     end
0158     obj.nsecs = val;
0159 end
0160 end
0161
0162 %%%%%%%%%%% Constructor %%%%%%%%%%%
0163 %
0164 %%%%%%%%%%% Constructor %%%%%%%%%%%
0165
0166 methods
0167     function obj = tsdata(varargin)
0168
0169         % Call data2D constructor since we add no special features here.
0170         obj = obj@data2D(varargin{:});
0171
0172         if nargin == 0
0173             %%%%%%%%%%% no inputs %%%%%%%%%%%
0174             %%%%%%%%%%% no inputs %%%%%%%%%%%
0175             %%%%%%%%%%% no inputs %%%%%%%%%%%
0176
0177         elseif nargin == 1
0178             %%%%%%%%%%% one input %%%%%%%%%%%
0179             %%%%%%%%%%% one input %%%%%%%%%%%
0180             %%%%%%%%%%% one input %%%%%%%%%%%
0181
0182             if isa(varargin{1}, 'tsdata')
0183                 %%%%%%%%%%% data = tsdata(tsdata-object) %%%%%%%%%%%
0184                 %----- Copy tsdata Object %%%%%%%%%%%
0185                 obj = copy(varargin{1}, 1);
0186
0187             elseif isstruct(varargin{1})
0188                 %%%%%%%%%%% data = tsdata(struct) %%%%%%%%%%%
0189                 %----- struct
0190
0191                 obj.t0 = utils.helper.struct2obj(varargin{1}.t0, 'time');
0192                 obj.fs = varargin{1}.fs;
0193                 obj.nsecs = varargin{1}.nsecs;
0194                 obj.version = varargin{1}.version;
0195
0196             elseif isnumeric(varargin{1})
0197                 %%%%%%%%%%% data = tsdata(y-vector) %%%%%%%%%%%
0198                 %----- y vector
0199                 obj.setY(varargin{1});
0200                 obj.fs = 1;
0201
0202             else
0203                 error('### Unknown single argument constructor.');
```



```

0239         obj.fs = 1;
0240         obj.t0 = varargin{2};
0241     else
0242         error('### Unknown two argument constructor.');
```

```

0243     end
0244     elseif nargin == 3
0245         %%%%%%%%%%% three input %%%%%%%%%%%
0246         %%%%%%%%%%% three input %%%%%%%%%%%
0247         %%%%%%%%%%% three input %%%%%%%%%%%
0248
0249         if numel(varargin{1}) == numel(varargin{2}) && numel(varargin{3}) == 1 &&
0250 ~isa(varargin{3}, 'time')
0251             %%%%%%%%%%% data = tsdata(x-vector, y-vector, fs) %%%%%%%%%%%
0252             % tsdata(x,y,fs)
0253             obj.setXY(varargin{1}, varargin{2});
0254             obj.fs = varargin{3};
0255
0256         elseif numel(varargin{1}) == numel(varargin{2}) && isa(varargin{3}, 'time')
0257             %%%%%%%%%%% data = tsdata(x-vector, y-vector, t0) %%%%%%%%%%%
0258             % tsdata(x,y,t0)
0259             [fs,t0,fitted] = tsdata.fitfs(varargin{1});
0260             obj.fs = fs;
0261             obj.t0 = varargin{3}+t0;
0262             if fitted
0263                 obj.setXY(varargin{1}, varargin{2});
0264             else
0265                 obj.setY(varargin{2});
0266             end
0267
0268         elseif numel(varargin{1}) > numel(varargin{2}) && isa(varargin{3}, 'time')
0269             %%%%%%%%%%% data = tsdata(y-vector, fs, t0) %%%%%%%%%%%
0270             % tsdata(y,fs,t0)
0271             obj.setY(varargin{1});
0272             obj.fs = varargin{2};
0273             obj.t0 = varargin{3};
0274         else
0275             error('### Unknown three argument constructor.');
```

```

0276     end
0277     elseif nargin == 4
0278         %%%%%%%%%%% four input %%%%%%%%%%%
0279         %%%%%%%%%%% four input %%%%%%%%%%%
0280         %%%%%%%%%%% four input %%%%%%%%%%%
0281
0282         if numel(varargin{1}) == numel(varargin{2}) && numel(varargin{3}) == 1 &&
0283 isa(varargin{4}, 'time')
0284             %%%%%%%%%%% data = tsdata(x-vector, y-vector, fs, t0) %%%%%%%%%%%
0285             [fs,t0,fitted] = tsdata.fitfs(varargin{1});
0286             if ~fitted
0287                 obj.setY(varargin{2});
0288             end
0289             obj.fs = fs;
0290             obj.t0 = varargin{4}+t0;
0291         else
0292             error('### Unknown four argument constructor.');
```

```

0293     end
0294     error('### Unknown number of constructor arguments.');
```

```

0295 end
0296
0297 % Now we can set nsecs
0298 if ~isempty(obj.x)
0299     % Then we have unevenly sampled data and the data duration
0300     % is taken as x(end) - x(1);
0301     obj.setNsecs(obj.x(end)-obj.x(1) + 1/obj.fs);
0302 else
0303     if ~isempty(obj.y)
0304         % Then the data is evenly sampled and the
0305         % duration of the data is easily computed.
0306         Ndata = length(obj.y);
0307         nsecs = Ndata / obj.fs;
0308         obj.setNsecs(nsecs);
0309     end
0310 end
0311 end % End constructor
0312 end % End public methods
0313
0314 %%%%%%%%%%% Methods (Public) %%%%%%%%%%%
0315 %
0316 %%%%%%%%%%% Methods (Public) %%%%%%%%%%%
0317
0318 methods
0319     % Setters
0320     varargout = setT0(varargin)
0321     varargout = setFs(varargin)
0322     varargout = setNsecs(varargin)

```



```
0323
0324 % Getters
0325 varargout = getX(varargin)
0326
0327 varargout = growT(varargin)
0328 varargout = fixNsecs(varargin)
0329 varargout = collapseX(varargin)
0330
0331 % Other public methods
0332 varargout = display(varargin);
0333 varargout = copy(varargin)
0334 % varargout = string(varargin);
0335 end
0336
0337 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0338 %                                     Methods (protected)                                     %
0339 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0340
0341 methods (Access = protected)
0342 end
0343
0344 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0345 %                                     Methods (private)                                     %
0346 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0347
0348 methods (Access = private)
0349 end
0350
0351 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0352 %                                     Methods (static)                                     %
0353 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0354 methods (Static)
0355     varargout = fitfs(varargin)
0356     varargout = update_struct(varargin);
0357
0358     function out = VEROUT()
0359         out = '$Id: tsdata.m,v 1.57 2008/09/03 16:39:37 hewitson Exp $';
0360     end
0361
0362     function ii = getInfo(varargin)
0363         ii = utils.helper.generic_getInfo(varargin{:}, 'tsdata');
0364     end
0365
0366     function out = SETS()
0367         out = {'Default'};
0368     end
0369
0370     function out = getDefaultPlist(set)
0371         switch set
0372         case 'Default'
0373             out = plist();
0374         otherwise
0375             error('### Unknown set [%s]', set);
0376         end
0377     end
0378 end % End static methods
0379
0380 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0381 %                                     Methods (static, private)                                     %
0382 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0383
0384 methods (Static, Access=private)
0385 end % End static, private methods
0386
0387 end % End classdef
0388
```

collapseX

PURPOSE ^

COLLAPSEX Checks whether the x vector is evenly sampled and then removes it

SYNOPSIS ^

function varargout = collapseX(varargin)

DESCRIPTION ^

COLLAPSEX Checks whether the x vector is evenly sampled and then removes it
DESCRIPTION: Checks whether the x vector is evenly sampled and then removes it after setting the t0 field.
CALL: obj = collapseX(obj)
INPUT: obj - tsdata object
OUTPUT: obj - tsdata object
VERSION: \$Id: collapseX.m,v 1.2 2008/09/04 15:29:32 ingo Exp \$
HISTORY: 12-06-2008 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of tsdata objects depending of the second input

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % COLLAPSEX Checks whether the x vector is evenly sampled and then removes it
0002 %
0003 %
0004 % DESCRIPTION: Checks whether the x vector is evenly sampled and then removes
0005 %               it after setting the t0 field.
0006 %
0007 % CALL:         obj = collapseX(obj)
0008 %
0009 % INPUT:        obj - tsdata object
0010 %
0011 % OUTPUT:       obj - tsdata object
0012 %
0013 % VERSION:      $Id: collapseX.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0014 %
0015 % HISTORY:      12-06-2008 M Hewitson
```

```
0016 %                               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 %
0020 function varargout = collapseX(varargin)
0021
0022 %% Check if this is a call for parameters
0023 if utils.helper.isinfocall(varargin{:})
0024     varargout{1} = getInfo(varargin{3});
0025     return
0026 end
0027
0028 obj = varargin{1};
0029
0030 %% decide whether we modify the pz-object, or create a new one.
0031 obj = copy(obj, nargout);
0032
0033 [fs,t0,fitted] = tsdata.fitfs(obj.x);
0034 if ~fitted
0035     % adjust T0
0036     obj.t0 = obj.t0 + obj.x(1);
0037     % adjust Nsecs
0038     obj.nsecs = length(obj.y)/obj.fs;
0039     % remove X samples
0040     obj.x = [];
0041 end
0042
0043 varargout{1} = obj;
0044 end
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %                               Local Functions
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 % FUNCTION:      getInfo
0053 %
0054 % DESCRIPTION:   Get Info Object
0055 %
0056 % HISTORY:       11-07-07 M Hewitson
0057 %                Creation.
0058 %
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl = [];
0065     else
0066         sets = {'Default'};
0067         pl = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id:
collapseX.m,v 1.2 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0071 end
0072
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getDefaultPlist
0076 %
0077 % DESCRIPTION:   Get Default Plist
0078 %
0079 % HISTORY:       11-07-07 M Hewitson
0080 %                Creation.
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 function plo = getDefaultPlist()
0085     plo = plist();
0086 end
0087
```

copy

PURPOSE ^

COPY Make copy of tsdata objects depending of the second input

SYNOPSIS ^

function varargout = **copy**(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of tsdata objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of tsdata objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> tsdata.getInfo('copy')

VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of tsdata objects depending of the second input
-  [tsdata](#) TSDATA time-series object class constructor.

This function is called by:

-  [collapseX](#) COLLAPSEX Checks whether the x vector is evenly sampled and then removes it
-  [copy](#) COPY Make copy of tsdata objects depending of the second input
-  [fixNsecs](#) FIXNSECS fixes the numer of seconds.
-  [growT](#) GROWT grows the time (x) vector if it is empty.
-  [setFs](#) SETFS Set the property 'fs'.
-  [setNsecs](#) SETNSECS Set the property 'nsecs'.
-  [setT0](#) SETT0 Set the property 't0'.
-  [tsdata](#) TSDATA time-series object class constructor.

SOURCE CODE ^

```
0001 % COPY Make copy of tsdata objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of tsdata objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> tsdata.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
```

```
0013 % HISTORY:      11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020
0021 if deepcopy
0022     s = size(old);
0023     obj(s(1),s(2)) = tsdata;
0024     for kk=1:numel(old)
0025         utils.helper.msg(utils.const.msg.OPROC2, 'copying tsdata');
0026         obj(kk).fs      = old(kk).fs;
0027         obj(kk).nsecs   = old(kk).nsecs;
0028         obj(kk).xunits  = old(kk).xunits;
0029         obj(kk).yunits  = old(kk).yunits;
0030         obj(kk).x       = old(kk).x;
0031         obj(kk).y       = old(kk).y;
0032         % copy time object
0033         obj(kk).t0      = copy(old(kk).t0, 1);
0034         % reset on copy
0035         obj(kk).version = tsdata.VEROUT;
0036     end
0037 else
0038     obj = old;
0039 end
0040 varargout{1} = obj;
0041 end
0042
```

fixNsecs

PURPOSE ^

FIXNSECS fixes the numer of seconds.

SYNOPSIS ^

function varargout = fixNsecs(varargin)

DESCRIPTION ^

```
FIXNSECS fixes the numer of seconds.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: fixes the numer of seconds.

CALL:      d      = fixNsecs(d)

INPUT:     d - tsdata object

OUTPUT:    d - tsdata object

VERSION:   $Id: fixNsecs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $

HISTORY:   12-06-2008 M Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of tsdata objects depending of the second input

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % FIXNSECS fixes the numer of seconds.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: fixes the numer of seconds.
0005 %
0006 % CALL:      d      = fixNsecs(d)
0007 %
0008 % INPUT:     d - tsdata object
0009 %
0010 % OUTPUT:    d - tsdata object
0011 %
0012 % VERSION:   $Id: fixNsecs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0013 %
0014 % HISTORY:   12-06-2008 M Hewitson
0015 %           Creation
0016 %
```

```
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 function varargout = fixNsecs(varargin)
0020
0021 %% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 obj = varargin{1};
0028
0029 %% decide whether we modify the pz-object, or create a new one.
0030 obj = copy(obj, nargout);
0031
0032 if isempty(obj.x)
0033     obj.nsecs = length(obj.y)/obj.fs;
0034 else
0035     obj.nsecs = obj.x(end)-obj.x(1) + 1/obj.fs;
0036 end
0037
0038 varargout{1} = obj;
0039 end
0040
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %                               Local Functions                               %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:      getInfo
0048 %
0049 % DESCRIPTION:   Get Info Object
0050 %
0051 % HISTORY:       11-07-07 M Hewitson
0052 %                Creation.
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id:
fixNsecs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:      getDefaultPlist
0071 %
0072 % DESCRIPTION:   Get Default Plist
0073 %
0074 % HISTORY:       11-07-07 M Hewitson
0075 %                Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082
```

getX

PURPOSE ^

GETX Get the property 'x'.

SYNOPSIS ^

function varargout = getX(varargin)

DESCRIPTION ^

```
GETX Get the property 'x'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the property 'x'.

CALL:      val = obj.getX();
           val = obj.getX(idx);
           val = obj.getX(1:10);

INPUTS:    obj - must be a single data2D object.
           idx - index of the data samples

M-FILE INFO: Get information about this methods by calling
             >> tsdata.getInfo('getX')

             Get information about a specified set-plist by calling:
             >> tsdata.getInfo('getX', 'None')

VERSION:   $Id: getX.m,v 1.4 2008/09/04 15:29:32 ingo Exp $

HISTORY:   27-05-2008 Diepholz
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETX Get the property 'x'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the property 'x'.
0005 %
0006 % CALL:      val = obj.getX();
0007 %           val = obj.getX(idx);
0008 %           val = obj.getX(1:10);
0009 %
0010 % INPUTS:    obj - must be a single data2D object.
0011 %           idx - index of the data samples
0012 %
```



```

0013 % M-FILE INFO: Get information about this methods by calling
0014 %      >> tsdata.getInfo('getX')
0015 %
0016 %      Get information about a specified set-plist by calling:
0017 %      >> tsdata.getInfo('getX', 'None')
0018 %
0019 % VERSION:      $Id: getX.m,v 1.4 2008/09/04 15:29:32 ingo Exp $
0020 %
0021 % HISTORY:      27-05-2008 Diepholz
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = getX(varargin)
0027
0028 %% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 obj = varargin{1};
0035
0036 idx = [];
0037 if nargin > 1
0038     if isnumeric(varargin{2}) || strcmpi(varargin{2}, 'end')
0039         idx = varargin{2};
0040     else
0041         error('### Unknown arguments to getX.');
```

```
0098 ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id: getX.m,v
1.4 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0099 end
0100
0101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0102 %
0103 % FUNCTION:      getDefaultPlist
0104 %
0105 % DESCRIPTION:  Get Default Plist
0106 %
0107 % HISTORY:      11-07-07 M Hewitson
0108 %                Creation.
0109 %
0110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111
0112 function plo = getDefaultPlist()
0113     plo = plist();
0114 end
0115
```

growT

PURPOSE ^

GROWT grows the time (x) vector if it is empty.

SYNOPSIS ^

function varargout = growT(varargin)

DESCRIPTION ^

```
GROWT grows the time (x) vector if it is empty.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: GROWT grows the time (x) vector if it is empty.

CALL:      d      = growT(d)

INPUT:     d - tsdata object

OUTPUT:    d - tsdata object

VERSION:   $Id: growT.m,v 1.3 2008/09/04 15:29:32 ingo Exp $

HISTORY:   12-06-2008 M Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of tsdata objects depending of the second input

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GROWT grows the time (x) vector if it is empty.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: GROWT grows the time (x) vector if it is empty.
0005 %
0006 % CALL:      d      = growT(d)
0007 %
0008 % INPUT:     d - tsdata object
0009 %
0010 % OUTPUT:    d - tsdata object
0011 %
0012 % VERSION:   $Id: growT.m,v 1.3 2008/09/04 15:29:32 ingo Exp $
0013 %
0014 % HISTORY:   12-06-2008 M Hewitson
0015 %           Creation
0016 %
```

```
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 function varargout = growT(varargin)
0020
0021 %% Check if this is a call for parameters
0022 if utils.helper.isinfocall(varargin{:})
0023     varargout{1} = getInfo(varargin{3});
0024     return
0025 end
0026
0027 d = varargin{1};
0028
0029 %% decide whether we modify the pz-object, or create a new one.
0030 d = copy(d, nargout);
0031
0032 % Grow the time vector
0033 d.setX(linspace(0, d.nsecs-1/d.fs, d.nsecs*d.fs));
0034
0035 varargout{1} = d;
0036 end
0037
0038 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0039 %                               Local Functions                               %
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043 %
0044 % FUNCTION:      getInfo
0045 %
0046 % DESCRIPTION:  Get Info Object
0047 %
0048 % HISTORY:      11-07-07 M Hewitson
0049 %               Creation.
0050 %
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052
0053 function ii = getInfo(varargin)
0054     if nargin == 1 && strcmpi(varargin{1}, 'None')
0055         sets = {};
0056         pl   = [];
0057     else
0058         sets = {'Default'};
0059         pl   = getDefaultPlist;
0060     end
0061     % Build info object
0062     ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id: growT.m,v
1.3 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0063 end
0064
0065 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0066 %
0067 % FUNCTION:      getDefaultPlist
0068 %
0069 % DESCRIPTION:  Get Default Plist
0070 %
0071 % HISTORY:      11-07-07 M Hewitson
0072 %               Creation.
0073 %
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075
0076 function plo = getDefaultPlist()
0077     plo = plist();
0078 end
0079
```

setFs

PURPOSE ^

SETFS Set the property 'fs'.

SYNOPSIS ^

function varargout = setFs(varargin)

DESCRIPTION ^

```
SETFS Set the property 'fs'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'fs'.

CALL:
    obj.setFs(12);
    obj = obj.setFs(12); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> tsdata.getInfo('setFs')

    Get information about a specified set-plist by calling:
    >> tsdata.getInfo('setFs', 'None')

VERSION:
    $Id: setFs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of tsdata objects depending of the second input

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETFS Set the property 'fs'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'fs'.
0005 %
0006 % CALL:
0007 %     obj.setFs(12);
0008 %     obj = obj.setFs(12); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> tsdata.getInfo('setFs')
0015 %
0016 %     Get information about a specified set-plist by calling:
0017 %     >> tsdata.getInfo('setFs', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setFs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %                >> tsdata.getInfo('setFs')
0013 %
0014 %                Get information about a specified set-plist by calling:
0015 %                >> tsdata.getInfo('setFs', 'None')
0016 %
0017 % VERSION:      $Id: setFs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%
0023 %
0024 function varargout = setFs(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'fs'
0039 obj.fs = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%
0045 %                Local Functions
0046 %%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id: setFs.m,v
1.2 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('fs', '');
0084 end
0085
```

setNsecs

PURPOSE ^

SETNSECS Set the property 'nsecs'.

SYNOPSIS ^

function varargout = setNsecs(varargin)

DESCRIPTION ^

```
SETNSECS Set the property 'nsecs'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'nsecs'.

CALL:
    obj.setNsecs(1000);
    obj = obj.setNsecs(1000); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> tsdata.getInfo('setNsecs')

    Get information about a specified set-plist by calling:
    >> tsdata.getInfo('setNsecs', 'None')

VERSION:
    $Id: setNsecs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of tsdata objects depending of the second input

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETNSECS Set the property 'nsecs'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'nsecs'.
0005 %
0006 % CALL:
0007 %     obj.setNsecs(1000);
0008 %     obj = obj.setNsecs(1000); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> tsdata.getInfo('setNsecs')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 %     >> tsdata.getInfo('setNsecs', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setNsecs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %           >> tsdata.getInfo('setNsecs')
0013 %
0014 %           Get information about a specified set-plist by calling:
0015 %           >> tsdata.getInfo('setNsecs', 'None')
0016 %
0017 % VERSION:      $Id: setNsecs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setNsecs(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 'nsecs'
0039 obj.nsecs = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                               Local Functions                               %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id:
setNsecs.m,v 1.2 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('nsecs', '');
0084 end
0085
```


setT0

PURPOSE ^

SETT0 Set the property 't0'.

SYNOPSIS ^

function varargout = setT0(varargin)

DESCRIPTION ^

```
SETT0 Set the property 't0'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 't0'.

CALL:
    obj.setT0(time(0));
    obj = obj.setT0(time(0)); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> tsdata.getInfo('setT0')

    Get information about a specified set-plist by calling:
    >> tsdata.getInfo('setT0', 'None')

VERSION:
    $Id: setT0.m,v 1.2 2008/09/04 15:29:32 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of tsdata objects depending of the second input

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETT0 Set the property 't0'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 't0'.
0005 %
0006 % CALL:
0007 %     obj.setT0(time(0));
0008 %     obj = obj.setT0(time(0)); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %     >> tsdata.getInfo('setT0')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 %     >> tsdata.getInfo('setT0', 'None')
0018 %
0019 % VERSION:
0020 %     $Id: setT0.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0021 %
0022 % HISTORY:
0023 %     27-05-2008 Diepholz
0024 %     Creation
0025 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0012 %                >> tsdata.getInfo('setT0')
0013 %
0014 %                Get information about a specified set-plist by calling:
0015 %                >> tsdata.getInfo('setT0', 'None')
0016 %
0017 % VERSION:      $Id: setT0.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setT0(varargin)
0025 %
0026 %%% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %%% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %%% set 't0'
0039 obj.t0 = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                Local Functions
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060     if nargin == 1 && strcmpi(varargin{1}, 'None')
0061         sets = {};
0062         pl = [];
0063     else
0064         sets = {'Default'};
0065         pl = getDefaultPlist;
0066     end
0067     % Build info object
0068     ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id: setT0.m,v
1.2 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083     plo = plist('t0', '');
0084 end
0085
```

display

PURPOSE ^

DISPLAY overloads display functionality for tsdata objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for tsdata objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for tsdata objects.

CALL:      txt      = display(tsdata)

INPUT:      tsdata - tsdta object

OUTPUT:     txt      - cell array with strings to display the tsdata object

M-FILE INFO: Get information about this methods by calling
              >> tsdata.getInfo('display')

              Get information about a specified set-plist by calling:
              >> tsdata.getInfo('display', 'None')

VERSION:    $Id: display.m,v 1.18 2008/09/04 15:29:32 ingo Exp $

HISTORY:     31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for tsdata objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for tsdata objects.
0005 %
0006 % CALL:      txt      = display(tsdata)
0007 %
0008 % INPUT:      tsdata - tsdta object
0009 %
0010 % OUTPUT:     txt      - cell array with strings to display the tsdata object
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> tsdata.getInfo('display')
```

```

0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> tsdata.getInfo('display', 'None')
0017 %
0018 % VERSION:      $Id: display.m,v 1.18 2008/09/04 15:29:32 ingo Exp $
0019 %
0020 % HISTORY:      31-01-2007 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027 %% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 tsdatas = utils.helper.collect_objects(varargin(:), 'tsdata');
0034
0035 txt = {};
0036
0037 for i=1:numel(tsdatas)
0038     ts = tsdatas(i);
0039     banner = sprintf('----- tsdata %02d -----', i);
0040     txt{end+1} = banner;
0041
0042     txt{end+1} = ' ';
0043
0044     xdata = ts.x;
0045     ydata = ts.y;
0046     yinfo = size(ydata);
0047     if ~isempty(xdata)
0048         xinfo = size(xdata);
0049     else
0050         xinfo = yinfo;
0051     end
0052
0053     txt{end+1} = sprintf('      fs:  %g', ts.fs);
0054     txt{end+1} = sprintf('      x:  [%d %d], %s', xinfo(1), xinfo(2), class(xdata));
0055     txt{end+1} = sprintf('      y:  [%d %d], %s', yinfo(1), yinfo(2), class(ydata));
0056     txt{end+1} = sprintf('xunits: %s', char(ts.xunits));
0057     txt{end+1} = sprintf('yunits: %s', char(ts.yunits));
0058     txt{end+1} = sprintf('nsecs:  %g', ts.nsecs);
0059     txt{end+1} = sprintf('t0:    %s', char(ts.t0));
0060
0061     banner_end(1:length(banner)) = '-';
0062     txt{end+1} = banner_end;
0063
0064     txt{end+1} = ' ';
0065 end
0066
0067 if nargin == 0
0068     for ii=1:length(txt)
0069         disp(txt{ii});
0070     end
0071 elseif nargin == 1
0072     varargout{1} = txt;
0073 end
0074
0075 end
0076
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %                               Local Functions                               %
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %
0083 % FUNCTION:      getInfo
0084 %
0085 % DESCRIPTION:   Get Info Object
0086 %
0087 % HISTORY:      11-07-07 M Hewitson
0088 %                Creation.
0089 %
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091
0092 function ii = getInfo(varargin)
0093 if nargin == 1 && strcmpi(varargin{1}, 'None')
0094     sets = {};
0095     pl = [];
0096 else
0097     sets = {'Default'};
0098     pl = getDefaultPlist;
0099 end

```

```
0100 % Build info object
0101 ii = minfo(mfilename, 'tsdata', '', utils.const.categories.output, '$Id: display.m,v
1.18 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0102 end
0103
0104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0105 %
0106 % FUNCTION:      getDefaultPlist
0107 %
0108 % DESCRIPTION:  Get Default Plist
0109 %
0110 % HISTORY:      11-07-07 M Hewitson
0111 %               Creation.
0112 %
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114
0115 function plo = getDefaultPlist()
0116     plo = plist();
0117 end
0118
```

xydata

PURPOSE ^

XYDATA X-Y data object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
XYDATA X-Y data object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: XYDATA X-Y data object class constructor.
             Create an X-Y data object.

SUPER CLASSES: data2D < ltpda_data < ltpda_nuo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
        version - cvs-version string
        xunits  - units of the y-axis
        yunits  - units of the y-axis
        x       - data values of the x-axis
        y       - data values of the y-axis

    Properties (read only)

Possible constructors:
    xy = xydata()      - creates a blank xy-data object
    xy = xydata(y)     - creates an xy data object with the given
                        y-data.
    xy = xydata(x,y)   - creates an xy-data object with the given
                        (x,y)-data.

XYDATA METHODS:

    Defined Abstract methods:
        char          - returns one character string which represents the object
        copy          - copies an object
        display       - displays an object
        update_struct - updates a object structure to the current tbx-version

M-FILE INFO: The following call returns an minfo object that contains
              information about the xydata constructor:
                >> info = xydata.getInfo
              or  >> info = xydata.getInfo('xydata')

              You can get information about class methods by calling:
                >> info = xydata.getInfo(method)
              e.g. >> info = xydata.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
                >> info = xydata.getInfo(method, set)
              e.g. >> info = xydata.getInfo('xydata', 'Default')

VERSION:  $Id: xydata.m,v 1.32 2008/09/03 16:56:21 ingo Exp $

HISTORY:  30-01-2007 Hewitson
           Creation

SEE ALSO: tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY Make copy of xydata objects depending of the second input
-  **display** DISPLAY overloads display functionality for xydata objects.
-  **update_struct** UPDATE_STRUCT update the input structure to the current ltpda version
-  **xydata** XYDATA X-Y data object class constructor.

This function is called by:

-  **copy** COPY Make copy of xydata objects depending of the second input
-  **xydata** XYDATA X-Y data object class constructor.

SUBFUNCTIONS ^

-  **function** obj = xydata(varargin)
-  **function** out = VEROUT()
-  **function** ii = getInfo(varargin)
-  **function** out = SETS()
-  **function** out = getDefaultPlist(set)

SOURCE CODE ^

```

0001 % XYDATA X-Y data object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: XYDATA X-Y data object class constructor.
0005 %             Create an X-Y data object.
0006 %
0007 % SUPER CLASSES: data2D < ltpda_data < ltpda_nuo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         version - cvs-version string
0013 %         xunits  - units of the y-axis
0014 %         yunits  - units of the y-axis
0015 %         x       - data values of the x-axis
0016 %         y       - data values of the y-axis
0017 %
0018 %     Properties (read only)
0019 %
0020 %     Possible constructors:
0021 %         xy = xydata()           - creates a blank xy-data object
0022 %         xy = xydata(y)         - creates an xy data object with the given
0023 %                                y-data.
0024 %         xy = xydata(x,y)       - creates an xy-data object with the given
0025 %                                (x,y)-data.
0026 %
0027 % XYDATA METHODS:
0028 %
0029 %     Defined Abstract methods:
0030 %         char           - returns one character string which represents the object
0031 %         copy           - copies an object
0032 %         display        - displays an object
0033 %         update_struct  - updates a object structure to the current tbx-version
0034 %
0035 % M-FILE INFO: The following call returns an minfo object that contains
0036 %              information about the xydata constructor:
0037 %              >> info = xydata.getInfo
0038 %              or  >> info = xydata.getInfo('xydata')
0039 %
0040 %              You can get information about class methods by calling:
0041 %              >> info = xydata.getInfo(method)
0042 %              e.g. >> info = xydata.getInfo('eq')
0043 %
0044 %              You can also restrict the sets of parameters contained in
0045 %              the minfo object by calling:
0046 %              >> info = xydata.getInfo(method, set)
0047 %              e.g. >> info = xydata.getInfo('xydata', 'Default')
0048 %
0049 % VERSION: $Id: xydata.m,v 1.32 2008/09/03 16:56:21 ingo Exp $
0050 %

```

```

0051 % HISTORY: 30-01-2007 Hewitson
0052 %          Creation
0053 %
0054 % SEE ALSO: tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata
0055 %
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 classdef xydata < data2D
0059
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061 %          Property definition %
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063
0064 %----- Public (read/write) Properties -----
0065 properties
0066 end
0067
0068 %----- Protected read-only Properties -----
0069 properties (SetAccess = protected)
0070     version = '$Id: xydata.m,v 1.32 2008/09/03 16:56:21 ingo Exp $';
0071 end
0072
0073 %----- Private Properties -----
0074 properties (GetAccess = protected, SetAccess = protected)
0075 end
0076
0077 %----- Abstract Properties -----
0078 properties (Abstract = true, SetAccess = protected)
0079 end
0080
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %          Check property setting %
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084
0085 methods
0086 end
0087
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089 %          Constructor %
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091
0092 methods
0093     function obj = xydata(varargin)
0094
0095         % Call data2D constructor since we add no special features here.
0096         obj = obj@data2D(varargin{:});
0097
0098         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0099         %          Set default values %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0100
0101         if nargin == 0
0102             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0103             %          no input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104
0105         elseif nargin == 1
0106             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0107             %          one input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109
0110             if isa(varargin{1}, 'xydata')
0111                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0112                 obj = xydata(xydata-object) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113                 obj = copy(varargin{1}, 1);
0114
0115             elseif isnumeric(varargin{1})
0116                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0117                 obj = xydata(y-vector) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0118                 obj.setY(varargin{1});
0119                 obj.setX(1:length(varargin{1}));
0120
0121             elseif isstruct(varargin{1})
0122                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0123                 data = xydata(struct) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0124                 obj.version = varargin{1}.version;
0125
0126                 %% Do nothing because all values are set in the super class
0127
0128             else
0129                 error('### Unknown single argument constructor.');
```



```
0137         end
0138
0139     else
0140         error('### Unknown two argument constructor.');
```

0141 end
0142 end % End constructor
0143 end
0144
0145 %%%
0146 % Methods (public) %
0147 %%%
0148
0149 methods
0150 end
0151
0152 %%%
0153 % Methods (public) %
0154 %%%
0155
0156 methods
0157 varargout = display(varargin);
0158 varargout = copy(varargin)
0159 end
0160
0161 %%%
0162 % Methods (protected) %
0163 %%%
0164
0165 methods (Access = protected)
0166 end
0167
0168 %%%
0169 % Methods (static) %
0170 %%%
0171
0172 methods (Static)
0173 varargout = update_struct(varargin);
0174
0175 function out = VEROUT()
0176 out = '\$Id: xydata.m,v 1.32 2008/09/03 16:56:21 ingo Exp \$';
0177 end
0178
0179 function ii = getInfo(varargin)
0180 ii = utils.helper.generic_getInfo(varargin{:}, 'xydata');
0181 end
0182
0183 function out = SETS()
0184 out = {'Default'};
0185 end
0186
0187 function out = getDefaultPlist(set)
0188 switch set
0189 case 'Default'
0190 out = plist();
0191 otherwise
0192 error('### Unknown set [%s]', set);
0193 end
0194 end
0195 end
0196
0197 %%%
0198 % Methods (abstract) %
0199 %%%
0200
0201 methods (Abstract)
0202 end
0203
0204 end
0205
0206

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

<http://www.lisa.aei-hannover.de/ltipda/usermanual/ug/classes/@xydata/xydata.html>[8/9/08 2:10:28 PM]

copy

PURPOSE ^

COPY Make copy of xydata objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

```
COPY Make copy of xydata objects depending of the second input
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COPY Make copy of xydata objects depending of the second input

CALL:      objs = copy(objs, [1,0]);

M-FILE INFO: Get information about this methods by calling
              >> xydata.getInfo('copy')

VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $

HISTORY:    11-07-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [xydata](#) XYDATA X-Y data object class constructor.

This function is called by:

-  [xydata](#) XYDATA X-Y data object class constructor.

SOURCE CODE ^

```
0001 % COPY Make copy of xydata objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of xydata objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> xydata.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.7 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('xydata');
0022         s = size(old);
0023         obj(s(1),s(2)) = xydata;
0024         for kk=1:numel(old)
0025             utils.helper.msg(utils.const.msg.OPROC2, 'copying xydata');
```

```
0026     for jj=1:length(ps)
0027         obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028     end
0029     % reset on copy
0030     obj(kk).version = xydata.VEROUT;
0031 end
0032 else
0033     obj = old;
0034 end
0035 varargout{1} = obj;
0036 end
0037
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

display

PURPOSE ^

DISPLAY overloads display functionality for xydata objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for xydata objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for xydata objects.

CALL:      txt      = display(xy)

INPUT:     xy - xydata object

OUTPUT:    txt      - cell array with strings to display the xydata object

M-FILE INFO: Get information about this methods by calling
              >> xydata.getInfo('display')

              Get information about a specified set-plist by calling:
              >> xydata.getInfo('display', 'None')

VERSION:   $Id: display.m,v 1.12 2008/09/04 15:29:32 ingo Exp $

HISTORY:   31-01-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [xydata](#) XYDATA X-Y data object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for xydata objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for xydata objects.
0005 %
0006 % CALL:      txt      = display(xy)
0007 %
0008 % INPUT:     xy - xydata object
0009 %
0010 % OUTPUT:    txt      - cell array with strings to display the xydata object
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %              >> xydata.getInfo('display')
```

```

0014 %
0015 %           Get information about a specified set-plist by calling:
0016 %           >> xydata.getInfo('display', 'None')
0017 %
0018 % VERSION:      $Id: display.m,v 1.12 2008/09/04 15:29:32 ingo Exp $
0019 %
0020 % HISTORY:      31-01-2007 M Hewitson
0021 %               Creation
0022 %
0023 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027     %% Check if this is a call for parameters
0028     if utils.helper.isinfocall(varargin{:})
0029         varargout{1} = getInfo(varargin{3});
0030         return
0031     end
0032
0033     xydatas = utils.helper.collect_objects(varargin(:), 'xydata');
0034
0035     txt = {};
0036
0037     for i=1:numel(xydatas)
0038         xy = xydatas(i);
0039         banner = sprintf('----- xydata %02d -----', i);
0040         txt{end+1} = banner;
0041
0042         txt{end+1} = ' ';
0043
0044         xdata = xy.x;
0045         ydata = xy.y;
0046         xinfo = size(xdata);
0047         yinfo = size(ydata);
0048
0049         txt{end+1} = sprintf('      x:  [%d %d], %s', xinfo(1), xinfo(2), class(xdata));
0050         txt{end+1} = sprintf('      y:  [%d %d], %s', yinfo(1), yinfo(2), class(ydata));
0051         txt{end+1} = sprintf('xunits: %s', char(xy.xunits));
0052         txt{end+1} = sprintf('yunits: %s', char(xy.yunits));
0053
0054         banner_end(1:length(banner)) = '-';
0055         txt{end+1} = banner_end;
0056
0057         txt{end+1} = ' ';
0058     end
0059
0060     if nargout == 0
0061         for ii=1:length(txt)
0062             disp(txt{ii});
0063         end
0064     elseif nargout == 1
0065         varargout{1} = txt;
0066     end
0067 end
0068
0069 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0070 %                               Local Functions                               %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getInfo
0076 %
0077 % DESCRIPTION:  Get Info Object
0078 %
0079 % HISTORY:      11-07-07 M Hewitson
0080 %               Creation.
0081 %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 function ii = getInfo(varargin)
0085     if nargin == 1 && strcmpi(varargin{1}, 'None')
0086         sets = {};
0087         pl = [];
0088     else
0089         sets = {'Default'};
0090         pl = getDefaultPlist;
0091     end
0092     % Build info object
0093     ii = minfo(mfilename, 'xydata', '', utils.const.categories.output, '$Id: display.m,v
1.12 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0094 end
0095
0096 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0097 %
0098 % FUNCTION:      getDefaultPlist

```

```
0099 %  
0100 % DESCRIPTION: Get Default Plist  
0101 %  
0102 % HISTORY:      11-07-07 M Hewitson  
0103 %              Creation.  
0104 %  
0105 %%%%%%%%%%%%%  
0106  
0107 function plo = getDefaultPlist()  
0108     plo = plist();  
0109 end  
0110
```

xyzdata

PURPOSE ^

XZYDATA X-Y-Z data object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
XZYDATA X-Y-Z data object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: XYZYDATA X-Y-Z data object class constructor.
             Create an X-Y-Z data object.

SUPER CLASSES: data3D < data2D < ltpda_data < ltpda_nuo < ltpda_obj

PROPERTIES:

    Inherit Properties (read only)
    version - cvs-version string
    xunits  - units of the x-axis
    yunits  - units of the y-axis
    zunits  - units of the z-axis
    x       - data values of the x-axis
    y       - data values of the y-axis
    z       - data values of the z-axis

Possible constructors:
    xyz = xyzdata()      - creates a blank xyz object
    xyz = xyzdata(z)     - creates an xyz object with the given
                          z-data.
    xyz = xydata(x,y,z)  - creates an xyz object with the given
                          (x,y,z)-data.

XYZYDATA METHODS:

    Defined Abstract methods:
    char      - returns one character string which represents the object
    copy      - copies an object
    display   - displays an object
    update_struct - updates a object structure to the current tbx-version

M-FILE INFO: The following call returns an minfo object that contains
              information about the xyzdata constructor:
              >> info = xyzdata.getInfo
              or  >> info = xyzdata.getInfo('xyzdata')

              You can get information about class methods by calling:
              >> info = xyzdata.getInfo(method)
              e.g. >> info = xyzdata.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
              >> info = xyzdata.getInfo(method, set)
              e.g. >> info = xyzdata.getInfo('xyzdata', 'Default')

VERSION:  $Id: xyzdata.m,v 1.12 2008/09/03 16:39:37 hewitson Exp $

HISTORY:  30-01-2007 Hewitson
           Creation

SEE ALSO: tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **copy** COPY Make copy of xyzdata objects depending of the second input
-  **display** DISPLAY overloads display functionality for xyzdata objects.
-  **update_struct** UPDATE_STRUCT update the input structure to the current ltpda version
-  **xyzdata** XZYDATA X-Y-Z data object class constructor.

This function is called by:

-  **copy** COPY Make copy of xyzdata objects depending of the second input
-  **xyzdata** XZYDATA X-Y-Z data object class constructor.

SUBFUNCTIONS [^]

-  **function** obj = xyzdata(varargin)
-  **function** out = VEROUT()
-  **function** ii = getInfo(varargin)
-  **function** out = SETS()
-  **function** out = getDefaultPlist(set)

SOURCE CODE [^]

```

0001 % XZYDATA X-Y-Z data object class constructor.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: XZYDATA X-Y-Z data object class constructor.
0005 %               Create an X-Y-Z data object.
0006 %
0007 % SUPER CLASSES: data3D < data2D < ltpda_data < ltpda_nuo < ltpda_obj
0008 %
0009 % PROPERTIES:
0010 %
0011 %     Inherit Properties (read only)
0012 %         version - cvs-version string
0013 %         xunits  - units of the x-axis
0014 %         yunits  - units of the y-axis
0015 %         zunits  - units of the z-axis
0016 %         x       - data values of the x-axis
0017 %         y       - data values of the y-axis
0018 %         z       - data values of the z-axis
0019 %
0020 %     Possible constructors:
0021 %         xyz = xyzdata()      - creates a blank xyz object
0022 %         xyz = xyzdata(z)    - creates an xyz object with the given
0023 %                               z-data.
0024 %         xyz = xydata(x,y,z) - creates an xyz object with the given
0025 %                               (x,y,z)-data.
0026 %
0027 % XYDATA METHODS:
0028 %
0029 %     Defined Abstract methods:
0030 %         char          - returns one character string which represents the object
0031 %         copy          - copies an object
0032 %         display       - displays an object
0033 %         update_struct - updates a object structure to the current tbx-version
0034 %
0035 % M-FILE INFO: The following call returns an minfo object that contains
0036 %               information about the xyzdata constructor:
0037 %               >> info = xyzdata.getInfo
0038 %               or   >> info = xyzdata.getInfo('xyzdata')
0039 %
0040 %               You can get information about class methods by calling:
0041 %               >> info = xyzdata.getInfo(method)
0042 %               e.g. >> info = xyzdata.getInfo('eq')
0043 %
0044 %               You can also restrict the sets of parameters contained in
0045 %               the minfo object by calling:
0046 %               >> info = xyzdata.getInfo(method, set)
0047 %               e.g. >> info = xyzdata.getInfo('xyzdata', 'Default')
0048 %
0049 % VERSION: $Id: xyzdata.m,v 1.12 2008/09/03 16:39:37 hewitson Exp $
0050 %

```



```

0051 % HISTORY: 30-01-2007 Hewitson
0052 %          Creation
0053 %
0054 % SEE ALSO: tsdata, fsdata, xydata, cdata, data2D, data3D, xyzdata
0055 %
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 classdef xyzdata < data3D
0059
0060 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061 %          Property definition %
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063
0064 %----- Public (read/write) Properties -----
0065 properties
0066 end
0067
0068 %----- Protected read-only Properties -----
0069 properties (SetAccess = protected)
0070     version = '$Id: xyzdata.m,v 1.12 2008/09/03 16:39:37 hewitson Exp $';
0071 end
0072
0073 %----- Private Properties -----
0074 properties (GetAccess = protected, SetAccess = protected)
0075 end
0076
0077 %----- Abstract Properties -----
0078 properties (Abstract = true, SetAccess = protected)
0079 end
0080
0081 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082 %          Check property setting %
0083 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084
0085 methods
0086 end
0087
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089 %          Constructor %
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091
0092 methods
0093     function obj = xyzdata(varargin)
0094
0095         % Call data2D constructor since we add no special features here.
0096         obj = obj@data3D(varargin{:});
0097
0098         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0099         %          Set default values %
0100         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0101
0102         if nargin == 0
0103             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0104             %          no inputs %
0105             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106
0107             elseif nargin == 1
0108                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0109                 %          one input %
0110                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111
0112                 if isa(varargin{1}, 'xyzdata')
0113                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114                     %          xy = xydata(xydata) %
0115                     obj = copy(varargin{1}, 1);
0116
0117                 elseif isstruct(varargin{1})
0118                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0119                     %          data = xyzdata(struct) %
0120                     obj.version = varargin{1}.version;
0121
0122                     %%% Do nothing. Because no properties are defined in this class.
0123
0124                 elseif isnumeric(varargin{1}) && length(varargin{1}) > 1
0125                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0126                     %          xyz = xyzdata(z_vector) %
0127                     obj.x = 1:size(varargin{1}, 1);
0128                     obj.y = 1:size(varargin{1}, 2);
0129                     obj.z = varargin{1};
0130
0131                 elseif isa(varargin{1}, 'plist')
0132                     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133                     %          ts = xydata(plist) %
0134                     if nparams(varargin{1}) == 0
0135                         % return empty object
0136                     else
0137                         error('### Unknown xyzdata constructor method.');

```

```
0137
0138     elseif nargin == 3
0139         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0140         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% three input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0142
0143         obj.x = varargin{1};
0144         obj.y = varargin{2};
0145         obj.z = varargin{3};
0146     else
0147         error('### Unknown number of constructor arguments.');
```

0148 end

0149

0150 %%% Normalize the the x- and y-vector %%%

0151

0152 nx = size(obj.z, 2);

0153 ny = size(obj.z, 1);

0154

0155 if length(obj.x) ~= nx

0156 error('### X-vector has wrong length.');

0157 end

0158 if length(obj.y) ~= ny

0159 error('### Y-vector has wrong length.');

0160 end

0161

0162 end % End constructor

0163 end

0164

0165 %%%

0166 %%% Methods (public) %%%

0167 %%%

0168

0169 methods

0170 varargout = copy(varargin)

0171 end

0172

0173 %%%

0174 %%% Methods (protected) %%%

0175 %%%

0176

0177 methods (Access = protected)

0178 end

0179

0180 %%%

0181 %%% Methods (public) %%%

0182 %%%

0183

0184 methods

0185 varargout = display(varargin);

0186 end

0187

0188 %%%

0189 %%% Methods (static) %%%

0190 %%%

0191

0192 methods (Static)

0193 varargout = update_struct(varargin);

0194

0195 function out = VEROUT()

0196 out = '\$Id: xyzdata.m,v 1.12 2008/09/03 16:39:37 hewitson Exp \$';

0197 end

0198

0199 function ii = getInfo(varargin)

0200 ii = utils.helper.generic_getInfo(varargin{:}, 'xyzdata');

0201 end

0202

0203 function out = SETS()

0204 out = {'Default'};

0205 end

0206

0207 function out = getDefaultPlist(set)

0208 switch set

0209 case 'Default'

0210 out = plist();

0211 otherwise

0212 error('### Unknown set [%s]', set);

0213 end

0214 end

0215

0216 end

0217

0218 %%%

0219 %%% Methods (abstract) %%%

0220 %%%

0221

0222 methods (Abstract)

0223	end
0224	
0225	end
0226	

getZ

PURPOSE ^

GETZ Get the property 'z'.

SYNOPSIS ^

function varargout = getZ(varargin)

DESCRIPTION ^

```
GETZ Get the property 'z'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the property 'z'.

CALL:      val = obj.getZ();
           val = obj.getZ(idx);
           val = obj.getZ(1:10);

INPUTS:    obj - must be a single data2D object.
           idx - index of the data samples

M-FILE INFO: Get information about this methods by calling
             >> fsdata.getInfo('getZ')

             Get information about a specified set-plist by calling:
             >> fsdata.getInfo('getZ', 'None')

VERSION:   $Id: getZ.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:   27-05-2008 Diepholz
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data3D](#) DATA3D is the abstract base class for 3-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % GETZ Get the property 'z'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the property 'z'.
0005 %
0006 % CALL:      val = obj.getZ();
0007 %           val = obj.getZ(idx);
0008 %           val = obj.getZ(1:10);
0009 %
0010 % INPUTS:    obj - must be a single data2D object.
0011 %           idx - index of the data samples
0012 %
```

```
0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> fsdata.getInfo('getZ')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> fsdata.getInfo('getZ', 'None')
0018 %
0019 % VERSION: $Id: getZ.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY: 27-05-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = getZ(varargin)
0027
0028 %%% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 %%% get idx
0035 idx = [];
0036 if nargin == 2
0037     if isnumeric(varargin{2})
0038         idx = varargin{2};
0039     else
0040         error('### Unknown arguments to getZ.');
```

0041 end
0042 end
0043
0044 %%% get z values
0045 if isempty(idx)
0046 varargout{1} = varargin{1}.z;
0047 else
0048 varargout{1} = varargin{1}.z(idx);
0049 end
0050 end
0051
0052 %%%
0053 % Local Functions %
0054 %%%
0055
0056 %%%
0057 %
0058 % FUNCTION: getInfo
0059 %
0060 % DESCRIPTION: Get Info Object
0061 %
0062 % HISTORY: 11-07-07 M Hewitson
0063 % Creation.
0064 %
0065 %%%
0066
0067 function ii = getInfo(varargin)
0068 if nargin == 1 && strcmpi(varargin{1}, 'None')
0069 sets = {};
0070 pl = [];
0071 else
0072 sets = {'Default'};
0073 pl = getDefaultPlist;
0074 end
0075 % Build info object
0076 ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '\$Id: getZ.m,v
1.2 2008/09/04 15:29:30 ingo Exp \$', sets, pl);
0077 end
0078
0079 %%%
0080 %
0081 % FUNCTION: getDefaultPlist
0082 %
0083 % DESCRIPTION: Get Default Plist
0084 %
0085 % HISTORY: 11-07-07 M Hewitson
0086 % Creation.
0087 %
0088 %%%
0089
0090 function plo = getDefaultPlist()
0091 plo = plist();
0092 end
0093

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

<http://www.lisa.aei-hannover.de/ltipda/usermanual/ug/classes/@data3D/getZ.html>[8/9/08 2:10:59 PM]

setZ

PURPOSE ^

SETZ Set the property 'z'.

SYNOPSIS ^

function varargout = setZ(varargin)

DESCRIPTION ^

```
SETZ Set the property 'z'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'z'.

CALL:
    obj.setZ([1 2 3; 1 2 3]);
    obj = obj.setZ([1 2 3; 1 2 3]); create copy of the object

INPUTS:
    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setZ')

    Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setZ', 'None')

VERSION:
    $Id: setZ.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
    Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data3D](#) DATA3D is the abstract base class for 3-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETZ Set the property 'z'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'z'.
0005 %
0006 % CALL:
0007 %         obj.setZ([1 2 3; 1 2 3]);
0008 %         obj = obj.setZ([1 2 3; 1 2 3]); create copy of the object
0009 %
0010 % INPUTS:
0011 %         obj - must be a single data2D object.
0012 %
0013 % M-FILE INFO: Get information about this methods by calling
0014 %         >> fsdata.getInfo('setZ')
0015 %
0016 %         Get information about a specified set-plist by calling:
```

```

0015 %                >> fsdata.getInfo('setZ', 'None')
0016 %
0017 % VERSION:      $Id: setZ.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0018 %
0019 % HISTORY:      27-05-2008 Diepholz
0020 %                Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023 %
0024 function varargout = setZ(varargin)
0025 %
0026 %% Check if this is a call for parameters
0027 if utils.helper.isinfocall(varargin{:})
0028     varargout{1} = getInfo(varargin{3});
0029     return
0030 end
0031 %
0032 obj = varargin{1};
0033 val = varargin{2};
0034 %
0035 %% decide whether we modify the pz-object, or create a new one.
0036 obj = copy(obj, nargout);
0037 %
0038 %% set 'z'
0039 obj.z = val;
0040 %
0041 varargout{1} = obj;
0042 end
0043 %
0044 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0045 %                                Local Functions                                %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %
0050 % FUNCTION:      getInfo
0051 %
0052 % DESCRIPTION:   Get Info Object
0053 %
0054 % HISTORY:      11-07-07 M Hewitson
0055 %                Creation.
0056 %
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %
0059 function ii = getInfo(varargin)
0060 if nargin == 1 && strcmpi(varargin{1}, 'None')
0061     sets = {};
0062     pl = [];
0063 else
0064     sets = {'Default'};
0065     pl = getDefaultPlist;
0066 end
0067 % Build info object
0068 ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id: setZ.m,v
1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0069 end
0070 %
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 %
0073 % FUNCTION:      getDefaultPlist
0074 %
0075 % DESCRIPTION:   Get Default Plist
0076 %
0077 % HISTORY:      11-07-07 M Hewitson
0078 %                Creation.
0079 %
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %
0082 function plo = getDefaultPlist()
0083 plo = plist('z', '');
0084 end
0085 %

```

setZunits

PURPOSE ^

SETZUNITS Set the property 'zunits'.

SYNOPSIS ^

function varargout = setZunits(varargin)

DESCRIPTION ^

```
SETZUNITS Set the property 'zunits'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Set the property 'zunits'.

CALL:
    obj.setZunits('units');
    obj = obj.setZunits('units'); create copy of the object

INPUTS:
    obj - must be a single data2D object.
    units - symbol-object or a valid string which can be transform
            into a symbolic-object.

M-FILE INFO: Get information about this methods by calling
    >> fsdata.getInfo('setZunits')

            Get information about a specified set-plist by calling:
    >> fsdata.getInfo('setZunits', 'None')

VERSION:
    $Id: setZunits.m,v 1.2 2008/09/04 15:29:30 ingo Exp $

HISTORY:
    27-05-2008 Diepholz
                Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [data3D](#) DATA3D is the abstract base class for 3-dimensional data objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % SETZUNITS Set the property 'zunits'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Set the property 'zunits'.
0005 %
0006 % CALL:
0007 %     obj.setZunits('units');
0008 %     obj = obj.setZunits('units'); create copy of the object
0009 %
0010 % INPUTS:
0011 %     obj - must be a single data2D object.
0012 %     units - symbol-object or a valid string which can be transform
               into a symbolic-object.
```



```
0013 % M-FILE INFO: Get information about this methods by calling
0014 % >> fsdata.getInfo('setZunits')
0015 %
0016 % Get information about a specified set-plist by calling:
0017 % >> fsdata.getInfo('setZunits', 'None')
0018 %
0019 % VERSION: $Id: setZunits.m,v 1.2 2008/09/04 15:29:30 ingo Exp $
0020 %
0021 % HISTORY: 27-05-2008 Diepholz
0022 % Creation
0023 %
0024 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0025
0026 function varargout = setZunits(varargin)
0027
0028 %% Check if this is a call for parameters
0029 if utils.helper.isinfocall(varargin{:})
0030     varargout{1} = getInfo(varargin{3});
0031     return
0032 end
0033
0034 obj = varargin{1};
0035 val = varargin{2};
0036
0037 %% decide whether we modify the pz-object, or create a new one.
0038 obj = copy(obj, nargout);
0039
0040 %% set 'zunits'
0041 if ischar(val)
0042     obj.zunits = sym(val);
0043 else
0044     obj.zunits = val;
0045 end
0046
0047 varargout{1} = obj;
0048 end
0049
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 % Local Functions
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %
0056 % FUNCTION: getInfo
0057 %
0058 % DESCRIPTION: Get Info Object
0059 %
0060 % HISTORY: 11-07-07 M Hewitson
0061 % Creation.
0062 %
0063 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064
0065 function ii = getInfo(varargin)
0066     if nargin == 1 && strcmpi(varargin{1}, 'None')
0067         sets = {};
0068         pl = [];
0069     else
0070         sets = {'Default'};
0071         pl = getDefaultPlist;
0072     end
0073     % Build info object
0074     ii = minfo(mfilename, 'data2D', '', utils.const.categories.internal, '$Id:
setZunits.m,v 1.2 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0075 end
0076
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %
0079 % FUNCTION: getDefaultPlist
0080 %
0081 % DESCRIPTION: Get Default Plist
0082 %
0083 % HISTORY: 11-07-07 M Hewitson
0084 % Creation.
0085 %
0086 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0087
0088 function plo = getDefaultPlist()
0089     plo = plist('zunits', sym('empty'));
0090 end
0091
```

copy

PURPOSE ^

COPY Make copy of xyzdata objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

COPY Make copy of xyzdata objects depending of the second input
DESCRIPTION: COPY Make copy of xyzdata objects depending of the second input
CALL: objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
>> xyzdata.getInfo('copy')
VERSION: \$Id: copy.m,v 1.6 2008/09/03 16:29:33 hewitson Exp \$
HISTORY: 11-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 xyzdata XZYDATA X-Y-Z data object class constructor.

This function is called by:

 xyzdata XZYDATA X-Y-Z data object class constructor.

SOURCE CODE ^

```
0001 % COPY Make copy of xyzdata objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of xyzdata objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> xyzdata.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.6 2008/09/03 16:29:33 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('xyzdata');
0022         s = size(old);
0023         obj(s(1),s(2)) = xyzdata;
0024         for kk=1:numel(old)
0025             utils.helper.msg(utils.const.msg.OPROC2, 'copying xyzdata');
```

```
0026     for jj=1:length(ps)
0027         obj(kk).(ps{jj}) = old(kk).(ps{jj});
0028     end
0029     % reset on copy
0030     obj(kk).version = xyzdata.VEROUT;
0031 end
0032 else
0033     obj = old;
0034 end
0035 varargout{1} = obj;
0036 end
0037
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

char

PURPOSE ^

CHAR convert a ltpda_data-object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

CHAR convert a ltpda_data-object into a string.
%%%

DESCRIPTION: CHAR convert a ltpda_data object into a string.

CALL: string = char(fsdata)

M-FILE INFO: Get information about this methods by calling
>> fsdata.getInfo('char')

Get information about a specified set-plist by calling:
>> fsdata.getInfo('char', 'set')

VERSION: \$Id: char.m,v 1.2 2008/09/04 15:29:32 ingo Exp \$

HISTORY: 23-07-2007 Diepholz
Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [display](#) DISPLAY overloads display functionality for xyzdata objects.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % CHAR convert a ltpda_data-object into a string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: CHAR convert a ltpda_data object into a string.
0005 %
0006 % CALL: string = char(fsdata)
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 % >> fsdata.getInfo('char')
0010 %
0011 % Get information about a specified set-plist by calling:
0012 % >> fsdata.getInfo('char', 'set')
0013 %
0014 % VERSION: $Id: char.m,v 1.2 2008/09/04 15:29:32 ingo Exp $
0015 %
0016 % HISTORY: 23-07-2007 Diepholz
0017 % Creation
```

```

0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = char(varargin)
0022
0023     %% Check if this is a call for parameters
0024     if utils.helper.isinfocall(varargin{:})
0025         varargout{1} = getInfo(varargin{3});
0026         return
0027     end
0028
0029     % Collect all ltpda_data-objects
0030     objs = utils.helper.collect_objects(varargin(:), 'xyzdata');
0031
0032
0033     %% Add the size of the data (all)
0034     pstr = sprintf('Ndata = %s, %s, [%sx%s]', num2str(length(objs.x)),
num2str(length(objs.y)), num2str(size(objs.z,1)), num2str(size(objs.z,2)));
0035
0036     varargout{1} = pstr;
0037 end
0038
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040 %                                     Local Functions                                     %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044 %
0045 % FUNCTION:      getInfo
0046 %
0047 % DESCRIPTION:   Get Info Object
0048 %
0049 % HISTORY:       11-07-07 M Hewitson
0050 %                 Creation.
0051 %
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053
0054 function ii = getInfo(varargin)
0055     if nargin == 1 && strcmpi(varargin{1}, 'None')
0056         sets = {};
0057         pl    = [];
0058     else
0059         sets = {'Default'};
0060         pl    = getDefaultPlist;
0061     end
0062     % Build info object
0063     ii = minfo(mfilename, 'xyzdata', '', utils.const.categories.output, '$Id: char.m,v 1.2
2008/09/04 15:29:32 ingo Exp $', sets, pl);
0064 end
0065
0066 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0067 %
0068 % FUNCTION:      getDefaultPlist
0069 %
0070 % DESCRIPTION:   Get Default Plist
0071 %
0072 % HISTORY:       11-07-07 M Hewitson
0073 %                 Creation.
0074 %
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076
0077 function plo = getDefaultPlist()
0078     plo = plist();
0079 end
0080

```

display

PURPOSE ^

DISPLAY overloads display functionality for xyzdata objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY overloads display functionality for xyzdata objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY overloads display functionality for xyzdata objects.

CALL:      txt      = display(xyzdata)

INPUT:     xyzdata - an xyz data object

OUTPUT:    txt      - cell array with strings to display the xyzdata object

M-FILE INFO: Get information about this methods by calling
              >> xyzdata.getInfo('display')

              Get information about a specified set-plist by calling:
              >> xyzdata.getInfo('display', 'None')

VERSION:    $Id: display.m,v 1.7 2008/09/04 15:29:32 ingo Exp $

HISTORY:    24-12-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a ltpda_data-object into a string.

This function is called by:

 [xyzdata](#) XZYDATA X-Y-Z data object class constructor.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DISPLAY overloads display functionality for xyzdata objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY overloads display functionality for xyzdata objects.
0005 %
0006 % CALL:      txt      = display(xyzdata)
0007 %
0008 % INPUT:     xyzdata - an xyz data object
0009 %
0010 % OUTPUT:    txt      - cell array with strings to display the xyzdata object
```

```

0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %      >> xyzdata.getInfo('display')
0014 %
0015 %      Get information about a specified set-plist by calling:
0016 %      >> xyzdata.getInfo('display', 'None')
0017 %
0018 % VERSION:      $Id: display.m,v 1.7 2008/09/04 15:29:32 ingo Exp $
0019 %
0020 % HISTORY:      24-12-2007 M Hewitson
0021 %                Creation
0022 %
0023 %%%%%%%%%%%%%%%
0024
0025 function varargout = display(varargin)
0026
0027 %%% Check if this is a call for parameters
0028 if utils.helper.isinfocall(varargin{:})
0029     varargout{1} = getInfo(varargin{3});
0030     return
0031 end
0032
0033 xyzdatas = utils.helper.collect_objects(varargin(:), 'xyzdata');
0034
0035 txt = {};
0036
0037 for i=1:numel(xyzdatas)
0038     xyz = xyzdatas(i);
0039     banner = sprintf('----- xyzdata %02d -----', i);
0040     txt{end+1} = banner;
0041
0042     txt{end+1} = ' ';
0043
0044     xdata = xyz.x;
0045     ydata = xyz.y;
0046     zdata = xyz.z;
0047     xinfo = whos('xdata');
0048     yinfo = whos('ydata');
0049     zinfo = whos('zdata');
0050
0051     txt{end+1} = sprintf('      x:  [%d %d], %s', xinfo.size(1), xinfo.size(2),
xinfo.class);
0052     txt{end+1} = sprintf('      y:  [%d %d], %s', yinfo.size(1), yinfo.size(2),
yinfo.class);
0053     txt{end+1} = sprintf('      z:  [%d %d], %s', zinfo.size(1), zinfo.size(2),
zinfo.class);
0054     txt{end+1} = sprintf('xunits:  %s', char(xyz.xunits));
0055     txt{end+1} = sprintf('yunits:  %s', char(xyz.yunits));
0056     txt{end+1} = sprintf('zunits:  %s', char(xyz.zunits));
0057
0058     banner_end(1:length(banner)) = '-';
0059     txt{end+1} = banner_end;
0060
0061     txt{end+1} = ' ';
0062 end
0063
0064 if nargout == 0
0065     for ii=1:length(txt)
0066         disp(txt{ii});
0067     end
0068 elseif nargout == 1
0069     varargout{1} = txt;
0070 end
0071
0072 end
0073
0074 %%%%%%%%%%%%%%%
0075 %                                Local Functions                                %
0076 %%%%%%%%%%%%%%%
0077
0078 %%%%%%%%%%%%%%%
0079 %
0080 % FUNCTION:      getInfo
0081 %
0082 % DESCRIPTION:  Get Info Object
0083 %
0084 % HISTORY:      11-07-07 M Hewitson
0085 %                Creation.
0086 %
0087 %%%%%%%%%%%%%%%
0088
0089 function ii = getInfo(varargin)
0090     if nargin == 1 && strcmpi(varargin{1}, 'None')
0091         sets = {};
0092         pl = [];
0093     else

```

```
0094     sets = {'Default'};
0095     pl     = getDefaultPlist;
0096 end
0097 % Build info object
0098 ii = minfo(mfilename, 'xyzdata', '', utils.const.categories.output, '$Id: display.m,v
1.7 2008/09/04 15:29:32 ingo Exp $', sets, pl);
0099 end
0100
0101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0102 %
0103 % FUNCTION:      getDefaultPlist
0104 %
0105 % DESCRIPTION:  Get Default Plist
0106 %
0107 % HISTORY:      11-07-07 M Hewitson
0108 %                Creation.
0109 %
0110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111
0112 function plo = getDefaultPlist()
0113     plo = plist();
0114 end
0115
```


Index for classes/@ao

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: ao %%%%%%%%%%%%%%
 abs	ABS overloads the Absolute value method for Analysis objects.
 acos	ACOS overloads the acos method for Analysis objects.
 ao	AO analysis object class constructor.
 ao2m	AO2M converts an analysis object to an '.m' file based on the history.
 applymethod	APPLYMETHOD to the analysis object
 applyoperator	APPLYOPERATOR to the analysis object
 asin	ASIN overloads the asin method for Analysis objects.
 atan	ATAN overloads the atan method for Analysis objects.
 atan2	ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse tangent.
 attachm	ATTACHM attach an m file to the analysis object.
 attachmdl	ATTACHMDL attach an mdl file to the analysis object.
 cat	CAT concatenate AOs into a vector.
 char	CHAR overloads char() function for analysis objects.
 cohere	COHERE makes coherence estimates of the time-series objects
 complex	COMPLEX overloads the complex operator for Analysis objects.
 compute	COMPUTE performs the given operations on the input AOs.
 computeDFT	COMPUTEDFT Computes DFT using FFT or Goertzel
 computeperiodogram	COMPUTEPERIODOGRAM Periodogram spectral estimation.
 conj	CONJ overloads the conjugate operator for Analysis objects.
 consolidate	CONSOLIDATE resamples all input AOs onto the same time grid.
 conv	CONV vector convolution.
 conv_noisegen	CONV_NOISEGEN calls the matlab function conv.m to convolute poles and zeros from a given pzmodel
 copy	COPY makes a (deep) copy of the input AOs.
 cos	COS overloads the cos operator for Analysis objects. Cosine of argument in radians.
 cpsd	CPSD makes cross-spectral density estimates of the time-series objects.
 ctranspose	CTRANSPOSE overloads the ' operator for Analysis Objects.
 delay	DELAY delays a time-series using various methods.
 demux	DEMUX splits the input vector of AOs into a number of output AOs.
 det	DET overloads the determinant function for Analysis objects.
 detrend	DETREND detrends the input analysis object using a polynomial of degree N.
 dft	DFT computes the DFT of the input time-series at the requested frequencies.
 diag	DIAG overloads the diagonal operator for Analysis Objects.
 diff	DIFF differentiates the data in AO.

 display	DISPLAY implement terminal display for analysis object.
 dopplercorr	FOO description for function 'foo' in one line. Necessary for lookfor functionality.
 downsample	DOWNSAMPLE AOs containing time-series data.
 dropduplicates	DROPDUPPLICATES drops all duplicate samples in time-series AOs.
 dsmean	DSMEAN performs a simple downsampling by taking the mean of every N samples.
 eig	EIG overloads the determinant function for Analysis objects.
 exp	EXP overloads the exp operator for Analysis objects. Exponential.
 export	EXPORT export an analysis object to a text file.
 extractm	EXTRACTM extracts an m-file from an analysis object and saves it to disk.
 extractmdl	EXTRACTMDL extracts an mdl file from an analysis object and saves it to disk.
 fft	FFT overloads the fft method for Analysis objects.
 filter	FILTER overrides the filter function for analysis objects.
 filtfilt	FILTFILT overrides the filtfilt function for analysis objects.
 find	FIND particular samples that satisfy the input query and return a new AO.
 findFsMax	findFsMax Returns the max Fs of a set of AOs
 findShortestVector	findShortestVector Returns the length of the shortest vector in samples
 firwhiten	FIRWHITEN whitens the input time-series by building an FIR whitening filter.
 fixfs	FIXFS resamples the input time-series to have a fixed sample rate.
 fngen	FNGEN creates an arbitrarily long time-series based on the input PSD.
 fq2fac	FQ2FAC is a private function and is called by ngconv.m which can be found in the
 fromDataInMAT	FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object
 fromDatafile	FROMDATAFILE Construct an ao from filename AND parameter list
 fromFSfcn	FROMFSFCN Construct an ao from a fs-function string
 fromFcn	FROMFCN Construct an ao from a function string
 fromPolyval	FROMPOLYVAL Construct a time-series ao from polynomial coefficients
 fromPzmodel	FROMPZMODEL Construct a time-series ao from polynomial coefficients
 fromSpecWin	FROMSPECWIN Construct an ao from a Spectral window
 fromTSfcn	FROMTSFCN Construct an ao from a ts-function string
 fromVals	FROMVALS Construct an ao from a value set
 fromWaveform	FROMWAVEFORM Construct an ao from a waveform
 fs	FS Get the data property 'fs'.
 gapfilling	GAPFILLING fills possible gaps in data.
 ge	GE overloads >= operator for analysis objects. Compare the y-axis values.
 gt	GT overloads > operator for analysis objects. Compare the y-axis values.
 hist	HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
 ifft	IFFT overloads the ifft operator for Analysis objects.
 imag	IMAG overloads the imaginary operator for Analysis objects.
 index	INDEX index into an AO array or matrix. This properly captures the history.
 interp	INTERP interpolate the values in the input AO(s) at new values.
 interpmissing	INTERPMISSING interpolate missing samples in a time-series.
 inv	INV overloads the inverse function for Analysis Objects.
 iplot	IPlot provides an intelligent plotting tool for LTPDA.

 join	JOIN multiple AOs into a single AO.
 lcohere	LCOHERE implement coherence estimation computed on a log frequency axis.
 lcpsd	LCPSD implement cross-power-spectral density estimation computed on a log frequency axis.
 le	LE overloads <= operator for analysis objects. Compare the y-axis values.
 len	LEN overloads the length operator for Analysis objects. Length of the data samples.
 lincom	LINCOM
 linedetect	LINEDETECT find spectral lines in the ao/fsdata objects.
 ln	LN overloads the log operator for Analysis objects. Natural logarithm.
 log	LOG overloads the log operator for Analysis objects. Natural logarithm.
 log10	LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.
 lpsd	LPSD implement the LPSD algorithm for analysis objects.
 lt	LT overloads < operator for analysis objects. Compare the y-axis values.
 ltf_plan	LTF_PLAN computes all input values needed for the LPSD and LTFE algorithms.
 ltfe	LTFE implement transfer-function estimation computed on a log frequency axis.
 lxspecc	LXSPEC performs log-scale cross-spectral analysis of various forms.
 max	MAX computes the maximum value of the data in the AO.
 mchol	MCHOL
 md5	MD5 computes an MD5 checksum from an analysis objects.
 mdc1_ifo2acc_fd	MDC1_IFO2ACC_FS calculates the external acceleration in the frequency-domain.
 mdc1_ifo2acc_inloop	MDC1_IFO2ACC_INLOOP calculates the inloop acceleration in the time-domain.
 mdc1_ifo2control	MDC1_IFO2CONTROL converts the input time-series to control forces.
 mdc1_input_noises	LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.
 mdc1_x2acc	MDC1_X2ACC converts the input time-series to acceleration with a time-domain filter
 mean	MEAN computes the mean value of the data in the AO.
 median	MEDIAN computes the median value of the data in the AO.
 min	MIN computes the minimum value of the data in the AO.
 minus	MINUS implements minus operator for analysis objects.
 mlpsd_m	MLPSD_M m-file only version of the LPSD algorithm
 mlpsd_mex	MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD
 mltfe	MLTFE compute log-frequency space TF
 mode	MODE computes the modal value of the data in the AO.
 mpower	MPOWER implements mpower operator for analysis objects.
 mrdivide	MRDIVIDE implements mrdivide operator for analysis objects.
 mtimes	MTIMES implements mtimes operator for analysis objects.
 ngconv	NGCONV is called by the function LTPDA_NOISEGEN
 nginit	NGINIT is called by the function LTPDA_NOISEGEN
 ngprop	NGPROP is called by the function LTPDA_NOISEGEN
 ngsetup	NGSETUP is called by the function LTPDA_NOISEGEN

 ngsetup_vpa	ALGONAME = mfilename;
 norm	NORM overloads the norm operator for Analysis Objects.
 phase	PHASE overloads the ltpda_phase operator for Analysis objects.
 plot	PLOT a simple plot of analysis objects.
 plus	PLUS implements addition operator for analysis objects.
 polyfit	POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
 power	POWER implements power operator for analysis objects.
 psd	PSD makes power spectral density estimates of the time-series objects
 pwelch	PWELCH makes power spectral density estimates of the time-series objects
 rdivide	RDIVIDE implements rdivide operator for analysis objects.
 real	REAL overloads the real operator for Analysis objects.
 resample	RESAMPLE overloads resample function for AOs.
 rms	RMS Calculate RMS deviation from spectrum
 search	SEARCH selects AOs that match the given name.
 select	SELECT select particular samples from the input AOs and return new AOs with only those samples.
 setDescription	SETDESCRIPTION sets the 'description' property of the ao.
 setFs	SETFS sets the 'fs' property of the ao.
 setT0	SETT0 sets the 't0' property of the ao.
 setX	SETX sets the 'x' property of the ao.
 setXY	SETXY sets the 'xy' property of the ao.
 setXunits	SETXUNITS sets the 'xunits' property of the ao.
 setY	SETY sets the 'y' property of the ao.
 setYunits	SETYUNITS sets the 'yunits' property of the ao.
 setZ	SETZ sets the 'z' property of the ao.
 sign	SIGN overloads the sign operator for Analysis objects.%
 sin	SIN overloads the sin method for Analysis objects.
 smoother	SMOOTHER smooths a given series of data points using the specified method.
 sort	SORT the values in the AO.
 spectrogram	SPECTROGRAM computes a spectrogram of the given ao/tsdata.
 spikecleaning	spikecleaning detects and corrects possible spikes in analysis objects
 split	SPLIT split an analysis object into the specified segments.
 sqrt	SQRT computes the square root of the data in the AO.
 std	STD computes the standard deviation of the data in the AO.
 string	STRING writes a command string that can be used to recreate the input Analysis object(s).
 sum	SUM computes the sum of the data in the AO.
 svd	SVD overloads the determinant function for Analysis objects.
 t0	T0 Get the data property 't0'.
 tan	TAN overloads the tan method for Analysis objects.
 tfe	TFE makes transfer function estimates of the time-series objects.
 timedomainfit	TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
 times	TIMES implements times operator for analysis objects.

 timeshift	TIMESHIFT for AO/tsdata objects, shifts the time axis such that $x(1) = 0$.
 transpose	TRANPOSE overloads the '.' operator for Analysis Objects.
 uminus	UMINUS overloads the uminus operator for Analysis objects.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version
 upsample	UPSAMPLE overloads upsample function for AOs.
 validate	VALIDATE checks that the input Analysis Object is reproducible and valid.
 var	VAR computes the variance of the data in the AO.
 welch	WELCH Welch spectral estimation method.
 welchparse	WELCHPARSE Parser for the PWELCH & SPECTROGRAM functions
 welchscale	WELCHSCALE scales the output of welch to be in the required units
 x	X Get the data property 'x'.
 xcorr	XCORR makes cross-correlation estimates of the time-series
 xspect	XSPEC performs cross-spectral analysis of various forms.
 xunits	XUNITS Get the data property 'xunits'.
 y	Y Get the data property 'y'.
 yunits	YUNITS Get the data property 'yunits'.
 zeropad	ZEROPAD zero pads the input data series.

Subsequent directories:

 [CVS](#)

Index for classes/@cdata

Matlab files in this directory:

-  [Contents](#) %%%%%%%%% class: cdata
%%%%%%%%%
-  [applymethod](#) APPLYMETHOD applys the given method to the input 2D data.
-  [applyoperator](#) APPLYOPERATOR applys the given operator to the two input data objects.
-  [cdata](#) CDATA is the constant data class.
-  [char](#) CHAR convert a cdata-object into a string.
-  [copy](#) COPY Make copy of cdata objects depending of the second input
-  [display](#) DISPLAY implement terminal display for cdata object.
-  [getY](#) GETY Get the property 'y'.
-  [setY](#) SETY Set the property 'y'.
-  [setYunits](#) SETYUNITS Set the property 'yunits'.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

-  [CVS](#)

Index for classes/@data2D

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: data2D
%%%%%%%%%%%%%
-  [applymethod](#) APPLYMETHOD applys the given method to the input 2D data.
-  [applyoperator](#) APPLYOPERATOR applys the given operator to the two input data objects.
-  [char](#) CHAR convert a ltpda_data-object into a string.
-  [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.
-  [getX](#) GETX Get the property 'x'.
-  [getY](#) GETY Get the property 'y'.
-  [setX](#) SETX Set the property 'x'.
-  [setXY](#) SETXY Set the property 'xy'.
-  [setXunits](#) SETXUNITS Set the property 'xunits'.
-  [setY](#) SETY Set the property 'y'.
-  [setYunits](#) SETYUNITS Set the property 'yunits'.

Subsequent directories:

 [CVS](#)

Index for classes/@data3D

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% class: data3D
%%%%%%%%%%%%%
-  [data3D](#) DATA3D is the abstract base class for 3-dimensional data objects.
-  [getZ](#) GETZ Get the property 'z'.
-  [setZ](#) SETZ Set the property 'z'.
-  [setZunits](#) SETZUNITS Set the property 'zunits'.

Subsequent directories:

-  CVS

Index for classes/@fsdata

Matlab files in this directory:

 Contents	%% class: fsdata %%
 copy	COPY Make copy of fsdata objects depending of the second input
 display	DISPLAY implement terminal display for fsdata object.
 fsdata	FSDATA frequency-series object class constructor.
 getFfromYFs	GETDSFROMYFS grows an evenly spaced frequency vector of N points for
 setEnbw	SETENBW Set the property 'enbw'.
 setFs	SETFS Set the property 'fs'.
 setNavs	SETNAVS Set the property 'navs'.
 setT0	SETT0 Set the property 't0'.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@history

Matlab files in this directory:

 Contents	%% class: history %%
 char	CHAR convert a param object into a string.
 copy	COPY Make copy of history objects depending of the second input
 display	DISPLAY implement terminal display for history object.
 getNodeNodes	GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
 getNodeNodes_plot	GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
 hist2dot	HIST2DOT converts a history object to a 'DOT' file suitable for processing with graphviz
 hist2m	HIST2M writes a new m-file that reproduces the analysis described in the history object.
 history	HISTORY History object class constructor.
 plot	PLOT plots a history object as a tree diagram.
 string	STRING writes a command string that can be used to recreate the input history object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@ltpda_data

Matlab files in this directory:

 [Contents](#) %%%%%%%%%%%%%%% class: ltpda_data
%%%%%%%%%%%%%

 [ltpda_data](#) LTPDA_DATA is the abstract base class for ltpda data objects.

Subsequent directories:

 CVS

Index for classes/@ltpda_filter

Matlab files in this directory:

-  [Contents](#) %%%%%%%%% class: ltpda_filter %%%%%%%%%
-  [ltpda_filter](#) LTPDA_FILTER is the abstract base class for ltpda filter objects.
-  [setA](#) SETA Set the property 'a'
-  [setFs](#) SETFS Set the property 'fs'
-  [setHistout](#) SETHISTOUT Set the property 'histout'
-  [setInfile](#) SETINFILE Set the property 'infile'

Subsequent directories:

-  [CVS](#)

Index for classes/@ltpda_nuo

Matlab files in this directory:

-  [Contents](#) %%%%%%%%% class: ltpda_nuo
%%%%%%%%%
-  [ltpda_nuo](#) LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.

Subsequent directories:

-  [CVS](#)

Index for classes/@ltpda_obj

Matlab files in this directory:

 Contents	%% class: ltpda_obj %%
 eq	EQ overloads the == operator for ltpda objects.
 isprop	ISPROP tests if the given field is one of the object properties.
 ltpda_obj	LTPDA_OBJ is the abstract ltpda base class.
 ne	NE overloads the ~= operator for ltpda objects.
 prependVersion	PREPENDVERSION Prepend the input-version to the property 'version'
 setVersion	SETVERSION Set the property 'version'

Subsequent directories:

 [CVS](#)

Index for classes/@ltpda_uo

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%% class: ltpda_uo
%%%%%%%%%
-  [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.
-  [retrieve](#) RETRIEVE retrieves a collection of objects from an LTPDA repository.
-  [save](#) SAVE overloads save operator for ltpda objects.
-  [setCreated](#) SETCREATED Set the property 'created'
-  [setCreator](#) SETCREATOR Set the property 'creator'
-  [setName](#) SETNAME Set the property 'name'.
-  [submit](#) SUBMIT submits the given collection of objects to an LTPDA Repository.

Subsequent directories:

 [CVS](#)

Index for classes/@ltpda_uoh

Matlab files in this directory:

 Contents	%% class: ltpda_uoh %%
 addHistory	ADDEHISTORY Add a history-object to the ltpda_uo object.
 fromDataInMAT	FROMDATAINMAT Default method to convert a data-array into an ltpda_uoh
 fromDatafile	FROMDATAFILE Default method to convert a data-file into an ltpda_uoh-object
 fromFile	Construct an ltpda_ob from a file
 fromLISO	FROMLISO Default method to read LISO files
 fromRepository	Retrieve an ltpda_uo from a repository
 ltpda_uoh	LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
 setHist	SETHIST Set the property 'hist'
 setName	SETNAME Set the property 'name'.
 type	TYPE converts the input objects to MATLAB functions.

Subsequent directories:

 [CVS](#)

Index for classes/@mfir

Matlab files in this directory:

 Contents	%% class: mfir %%
 char	CHAR convert a mfir object into a string.
 copy	COPY Make copy of mfir objects depending of the second input
 display	DISPLAY overloads display functionality for mfir objects.
 fromA	Construct an mfir from coefficients
 fromAO	create FIR filter from magnitude of input AO/fsdata
 fromPzmodel	Construct an mfir from a pzmodel
 fromStandard	Construct an mfir from a standard types
 mfir	MFIR FIR filter object class constructor.
 mkbandpass	MKBANDPASS return a bandpass filter mfir(). A Cheby filter is used.
 mkbandreject	MKBANDREJECT return a low pass filter mfir(). A Butterworth filter is used.
 mkhighpass	MKHIGHPASS return a high pass filter mfir(). A Butterworth filter is used.
 mklowpass	MKLOWPASS return a low pass filter mfir().
 parseFilterParams	PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard FIR filter.
 redesign	REDESIGN redesign the input filter to work for the given sample rate.
 resp	RESP Make a frequency response of the filter.
 setGd	SETGD Set the property 'gd'
 string	STRING writes a command string that can be used to recreate the input filter object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:



Index for classes/@miir

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: miir %%%%%%%%%%%%%%
 char	CHAR convert a miir object into a string.
 copy	COPY Make copy of miir objects depending of the second input
 display	DISPLAY overloads display functionality for miir objects.
 filload	Load a LISO *_iir.fil file to get the filter taps and return a
 fromAB	Construct an miir from coefficients
 fromLISO	FROMLISO Construct a miir filter from a LISO file
 fromPzmodel	Construct an miir from a pzmodel
 fromStandard	Construct an miir from a standard types
 miir	MIIR IIR filter object class constructor.
 mkbandpass	MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
 mkbandreject	MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.
 mkhighpass	MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.
 mklowpass	MKLOWPASS return a low pass filter miir(). A Butterworth filter is used.
 parseFilterParams	PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard IIR filter.
 redesign	REDESIGN redesign the input filter to work for the given sample rate.
 resp	RESP Make a frequency response of the filter.
 setB	SETB Set the property 'b'
 setHistin	SETHISTIN Set the property 'histin'
 string	STRING writes a command string that can be used to recreate the input filter object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@minfo

Matlab files in this directory:

 Contents	%% class: minfo %%
 char	CHAR convert an minfo object into a string.
 copy	COPY Make copy of minfo objects depending of the second input
 display	DISPLAY display an minfo object.
 minfo	MINFO a helper class for LTPDA methods.
 setMversion	SETMVERSION Set the property 'mversion'.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@param

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: param %%%%%%%%%%%%%%%%%%%%%%%%
 char	CHAR convert a param object into a string.
 copy	COPY Make copy of param objects depending of the second input
 display	DISPLAY display a parameter
 display2	DISPLAY display a parameter
 mux	MUX concatenate params into a vector.
 param	PARAM Parameter object class constructor.
 setKey	SETKEY Set the property 'key'.
 setKeyVal	SETKEYVAL Set the properties 'key' and 'val'
 setVal	SETVAL Set the property 'val'.
 string	STRING writes a command string that can be used to recreate the input param object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@plist

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: plist %%%%%%%%%%%%%%
 append	APPEND append a param-object, plist-object or a key/value pair to the parameter list.
 char	CHAR convert a parameter list into a string.
 combine	COMBINE multiple parameter lists (plist objects) into a single plist.
 copy	COPY Make copy of plist objects depending of the second input
 display	DISPLAY display plist object.
 find	FIND overloads find routine for a parameter list.
 fromFile	Construct a ltpda_ob from a file
 fromRepository	Construct a plist filter from a repository
 isparam	ISPARAM look for a given key in the parameter lists.
 nparams	NPARAMS returns the number of param objects in the list.
 plist	PLIST Plist class object constructor.
 pset	PSET set or add a key/value pair or a param-object into the parameter list.
 remove	REMOVE remove a parameter from the parameter list.
 resp	RESP shadows miir/iirResp and pzmodel/resp.
 string	STRING converts a plist object to a command string which will recreate the plist object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@provenance

Matlab files in this directory:

 Contents	%% class: provenance %%
 char	CHAR convert a provenance object into a string.
 copy	COPY Make copy of provenance objects depending of the second input
 display	DISPLAY overload terminal display for provenance objects.
 provenance	PROVENANCE constructors for provenance class.
 string	STRING writes a command string that can be used to recreate the input provenance object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@pz

Matlab files in this directory:

 Contents	%% class: pz %%%
 char	CHAR convert a pz object into a string.
 copy	COPY Make copy of pz objects depending of the second input
 cp2iir	CP2IIR Return a,b IIR filter coefficients for a complex pole designed using the bilinear transform.
 cz2iir	CZ2IIR return a,b IIR filter coefficients for a complex zero designed using the bilinear transform.
 display	DISPLAY display a pz object.
 fq2ri	FQ2RI Convert frequency/Q pole/zero representation
 pz	PZ is the ltpda class that provides a common definition of poles and zeros.
 resp	RESP returns the complex response of the pz object.
 ri2fq	RI2FQ Convert complex pole/zero into frequency/Q pole/zero representation.
 rp2iir	RP2IIR Return a,b coefficients for a real pole designed using the bilinear transform.
 rz2iir	RZ2IIR Return a,b IIR filter coefficients for a real zero designed using the bilinear transform.
 setF	SETF Set the property 'f'
 setQ	SETQ Set the property 'q'
 setRI	SETRI Set the property 'ri' and computes 'f' and 'q'
 string	STRING writes a command string that can be used to recreate the input pz object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:



CVS

Index for classes/@pzmodel

Matlab files in this directory:

 Contents	%% class: pzmodel %%%
 abccascade	ABCCASCADE Cascade two filters together to get a new filter.
 char	CHAR convert a pzmodel object into a string.
 copy	COPY Make copy of pzmodel objects depending of the second input
 display	DISPLAY overloads display functionality for pzmodel objects.
 fngen	FNGEN creates an arbitrarily long time-series based on the input pzmodel.
 fromLISO	FROMLISO Construct a pzmodel from a LISO file
 fromPolesAndZeros	%%
 getlowerFreq	GETLOWERFREQ gets the frequency of the lowest pole or zero in the model.
 getupperFreq	GETUPPERFREQ gets the frequency of the highest pole or zero in the model.
 pzm2ab	PZM2AB convert pzmodel to IIR filter coefficients using bilinear transform.
 pzmodel	PZMODEL constructor for pzmodel class.
 resp	RESP returns the complex response of a pzmodel as an Analysis Object.
 setGain	SETGAIN Set the property 'gain'
 setPoles	SETPOLES Set the property 'poles'
 setZeros	SETZEROS Set the property 'zeros'
 string	STRING writes a command string that can be used to recreate the input pzmodel object.
 tomfir	TOMFIR approximates a pole/zero model with an FIR filter.
 tomiir	TOMIIR converts a pzmodel to an IIR filter using a bilinear transform.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 CVS

Index for classes/@specwin

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: specwin %%%%%%%%%%%%%%
 char	CHAR convert a specwin object into a string.
 copy	COPY Make copy of specwin objects depending of the second input
 display	DISPLAY overloads display functionality for specwin objects.
 get_window	GET_WINDOW returns the required window function as a structure.
 kaiser_alpha	KAISER_ALPHA returns the alpha parameter that gives the required input
 kaiser_flatness	KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser
 kaiser_nenbw	KAISER_NENBW returns the normalized noise-equivalent bandwidth for a
 kaiser_rov	KAISER_ROV returns the recommended overlap for a Kaiser window with
 kaiser_w3db	KAISER_W3DB returns the 3dB bandwidth in bins of a kaiser window with
 plot	PLOT plots a specwin object.
 specwin	% SPECWIN spectral window object class constructor.
 string	STRING writes a command string that can be used to recreate the input window object.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version
 win_BH92	WIN_BH92 return BH92 window sample for input z.
 win_Bartlett	WIN_BARTLETT return Bartlett window sample for input z.
 win_FTHP	WIN_FTHP return FTHP window sample for input z.
 win_FTNl	WIN_FTNl return FTNI window sample for input z.
 win_FTSRS	WIN_FTSRS return FTSRS window sample for input z.
 win_HFT116D	WIN_HFT116D return HFT116D window sample for input z.
 win_HFT144D	WIN_HFT144D return HFT144D window sample for input z.
 win_HFT169D	WIN_HFT169D return HFT169D window sample for input z.
 win_HFT196D	WIN_HFT196D return HFT196D window sample for input z.
 win_HFT223D	WIN_HFT223D return HFT223D window sample for input z.
 win_HFT248D	WIN_HFT248D return HFT248D window sample for input z.
 win_HFT70	WIN_HFT70 return HFT70 window sample for input z.
 win_HFT90D	WIN_HFT90D return HFT90D window sample for input z.
 win_HFT95	WIN_HFT95 return HFT95 window sample for input z.
 win_Hamming	WIN_HAMMING return Hamming window sample for input z.
 win_Hanning	WIN_HANNING return Hanning window sample for input z.
 win_Nuttall3	WIN_NUTTALL3 return Nuttall3 window sample for input z.
 win_Nuttall3a	WIN_NUTTALL3A return Nuttall3a window sample for input z.
 win_Nuttall3b	WIN_NUTTALL3B return Nuttall3b window sample for input z.
 win_Nuttall4	WIN_NUTTALL4 return Nuttall4 window sample for input z.
win_Nuttall4a	WIN_NUTTALL4A return Nuttall4a window sample for input z.
win_Nuttall4b	WIN_NUTTALL4B return Nuttall4b window sample for input z.

-  [win_Nuttall4c](#) WIN_NUTTALL4C return Nuttall4c window sample for input z.
-  [win_Rectangular](#) WIN_RECTANGULAR return rectangular window sample for input z.
-  [win_SFT3F](#) WIN_SFT3F return SFT3F window sample for input z.
-  [win_SFT3M](#) WIN_SFT3M return SFT3M window sample for input z.
-  [win_SFT4F](#) WIN_SFT4F return SFT4F window sample for input z.
-  [win_SFT4M](#) WIN_SFT4M return SFT4M window sample for input z.
-  [win_SFT5F](#) WIN_SFT5F return SFT5F window sample for input z.
-  [win_SFT5M](#) WIN_SFT5M return SFT5M window sample for input z.
-  [win_Welch](#) WIN_WELCH return Welch window sample for input z.

Subsequent directories:

 CVS

Generated on Mon 08-Sep-2008 13:18:35 by [m2html](#) © 2003

Index for classes/@ssm

Matlab files in this directory:

 Contents	%%%%%%%%%% class: ssm %%%%%%%%%%
 assemble	assembles embedded subsystems, with exogenous inputs
 cell_add	adds corresponding matrices of same sizes or empty inside cell array
 cell_fusion	fusions a block defined matrix stored inside cell array into one matrix
 cell_mult	multiplies block defined matrix stored inside cell array
 cell_recut	cuts a matrix into blocks stored inside cell array
 cell_select	CELL_SELECT selects lines and columns of a block defined matrix stored in
 cellstrfind	cellstrfind finds matching strings inside a cellstr array
 char	CHAR convert a ssm object into a string.
 copy	COPY Make copy of ssm objects depending of the second input
 display	DISPLAY display ssm object.
 double	Convert a statespace model object to double arrays for given i/o
 isstable	tells if ssm is numerically stable
 kalman	kalman applies Kalman filtering to a discrete ssm with given i/o
 minreal	minreal gives a minimal realization of a ssm object by deleting state
 modifparams	modifparams enables to options(i_options)y and substitute parameters
 modiftimestep	modiftime modifies the timestep of a ssm object
 modify	modify allows to exectue a string to modify a ssm object
 reduce	reduce enables to do model simplification
 reduce_model	REDUCE_MODEL enables to do model simplification
 simulate	simulate simulates a discrete ssm with given inputs
 ssm	SSM statespace model class constructor.
 ssm2iirpz	ssm2iirpz converts a statespace model object to an miir or a pzmodel
 ssm2ss	SSM2SS converts a statespace model object to a MATLAB statespace object.
 ssmFromBuiltinSystem	%%%%%%%%%
 ssmFromDescription	%%%%%%%%%
 ssmFromFilename	%%%%%%%%%
 ssmFromMiir	%%%%%%%%%
 ssmFromPzmodel	%%%%%%%%%
 ssmFromRepository	%%%%%%%%%
 ssmFromss	%%%%%%%%%
 string	STRING writes a command string that can be used to recreate the input statespace model object.

-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version
-  [validate](#) %%

Subsequent directories:

-  CVS

Index for classes/@time

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: time %%%%%%%%%%%%%%
 char	CHAR convert a time object into a string.
 copy	COPY Make copy of time objects depending of the second input
 display	DISPLAY overloads display functionality for time objects.
 epoch2str	EPOCH2STR Returns computes the time sting of the epoch time stored in 'utc_epoch_milli' and 'created'.
 format	FORMAT Returns the time in specified format.
 fromTimeString	FROMTIMESTRING creates a time object from a time string.
 fromTimeValue	FROMTIMEVALUE creates a time object from unix epoch time.
 getTimezones	GETTIMEZONES Get all possible timezones.
 minus	MINUS overloads - operator for time objects.
 parse_time_string	PRASE_TIME_STRING parses the time string.
 plus	PLUS overloads + operator for time objects.
 setEpochtime	SETEPOCHTIME Set the property 'utc_epoch_milli'.
 setTime_str	SETTIME_STR Set the property 'time_str'.
 setTimeformat	SETTIMEFORMAT Set the property 'timeformat'.
 setTimezone	SETTIMEZONE Set the property 'timezone'.
 string	STRING writes a command string that can be used to recreate the input time object.
 time	TIME time object class constructor.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:



CVS

Index for classes/@timespan

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%% class: timespan %%%%%%%%%%%
-  [char](#) CHAR convert a timespan object into a string.
-  [computeInterval](#) COMPUTEINTERVAL compute the interval of the time span.
-  [copy](#) COPY Make copy of time objects depending of the second input
-  [display](#) DISPLAY overloads display functionality for timespan objects.
-  [fromTimespanDef](#) %%%%%%%%%%%
-  [setEndT](#) SETENDT Set the property 'endT'.
-  [setStartT](#) SETSTARTT Set the property 'startT'.
-  [setTimeformat](#) SETTIMEFORMAT Set the property 'timeformat'.
-  [setTimezone](#) SETTIMEZONE Set the property 'timezone'.
-  [string](#) STRING writes a command string that can be used to recreate the input timespan object.
-  [timespan](#) TIMESPAN timespan object class constructor.
-  [update_struct](#) UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

-  [CVS](#)

Index for classes/@tsdata

Matlab files in this directory:

 Contents	%% class: tsdata %%
 collapseX	COLLAPSEX Checks whether the x vector is evenly sampled and then removes it
 copy	COPY Make copy of tsdata objects depending of the second input
 display	DISPLAY overloads display functionality for tsdata objects.
 fitfs	FITFS computes the sample rate of the input data. If the min and max sample
 fixNsecs	FIXNSECS fixes the numer of seconds.
 getX	GETX Get the property 'x'.
 growT	GROWT grows the time (x) vector if it is empty.
 setFs	SETFS Set the property 'fs'.
 setNsecs	SETNSECS Set the property 'nsecs'.
 setT0	SETT0 Set the property 't0'.
 tsdata	TSDATA time-series object class constructor.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@unit

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%%%%%%%%%%%% class: unit %%%%%%%%%%%%%%%%%%%%%%%%
 char	
 copy	COPY Make copy of unit objects depending of the second input
 display	DISPLAY display an unit object.
 mpower	
 mrdivide	
 mtimes	
 plus	
 power	
 rdivide	
 sqrt	
 string	STRING converts a unit object to a command string which will recreate the unit object.
 times	
 unit	MINFO a helper class for implementing units in LTPDA.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version

Subsequent directories:

 [CVS](#)

Index for classes/@xydata

Matlab files in this directory:

 Contents	%% class: xydata %%
 copy	COPY Make copy of xydata objects depending of the second input
 display	DISPLAY overloads display functionality for xydata objects.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version
 xydata	XYDATA X-Y data object class constructor.

Subsequent directories:

 [CVS](#)

Index for classes/@xyzdata

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% class: xyzdata %%%%%%%%%%%%%%
 char	CHAR convert a ltpda_data-object into a string.
 copy	COPY Make copy of xyzdata objects depending of the second input
 display	DISPLAY overloads display functionality for xyzdata objects.
 update_struct	UPDATE_STRUCT update the input structure to the current ltpda version
 xyzdata	XZYDATA X-Y-Z data object class constructor.

Subsequent directories:

 [CVS](#)

Index for m/etc

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/etc
%%%%%%%%%%%%%
-  [ao_fcn_template](#) FOO description for function 'foo' in one line. Necessary for lookfor functionality.
-  [ltpda_startup](#) This is the startup file for ltpda. It should be run once in the MATLAB
-  [ltpda_user_fcn_template](#) LTPDA_FOO is the standard function template for user functions
-  [ltpda_versions](#)
-  [method_fcn_template](#) FOO is the standard function template for class methods.

Subsequent directories:

-  [CVS](#)

Index for m/gui/@jcontrol

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/gui/@jcontrol %%%%%%%%%%%%%%%
-  [ancestor](#) ANCESTOR function overloaded as JCONTROL method
-  [close](#) Close methods overloaded for JCONTROL objects
-  [delete](#) DELETE method overloaded for the JCONTROL class
-  [get](#) GET method overloaded for JCONTROL class
-  [getappdata](#) GETAPPDATA function overloaded for JCONTROL class
-  [isappdata](#) ISAPPDATA function overloaded for the JCONTROL class
-  [jcontrol](#) JCONTROL constructor for JCONTROL class
-  [rmappdata](#) RMAPPDATA function overloaded for the JCONTROL class
-  [set](#) SET method overloaded for JCONTROL class
-  [setappdata](#) SETAPPDATA function overloaded for JCONTROL class
-  [subsasgn](#) SUBSASGN method overloaded for JCONTROL class
-  [subsref](#) SUBSREF method overloaded for jcontrol class

Subsequent directories:

-  [CVS](#)
-  [private](#)

Index for m/gui/@jcontrol/private

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/gui/@jcontrol/private
%%%%%%%%%%%%%
-  [VisibleProperty](#) VisibleProperty - helper function for JCONTROL methods

Subsequent directories:

-  [CVS](#)

Index for m/gui/ao_browser

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/gui/ao_browser
%%%%%%%%%%%%%
-  [ltpda_explorer](#) LTPDA_EXPLORER for exploring analysis objects and plotting/displaying their fields
-  [ltpda_javacomponent](#) WARNING: This feature is not supported in MATLAB
-  [ltpda_uitree](#) WARNING: This feature is not supported in MATLAB
-  [ltpda_uitreenode](#) WARNING: This feature is not supported in MATLAB

Subsequent directories:

-  [CVS](#)
-  [exp_struct_icons](#)

Index for m/gui/constructor_helper

Matlab files in this directory:

 [Contents](#)

%% path:
m/gui/constructor_helper
%%

 [ltpda_constructor_helper](#)

LTPDA_CONSTRUCTOR_HELPER allows the user to explore object constructors.

 [ltpda_constructor_helper_build](#)

Callback for the build constructor gui

 [ltpda_constructor_helper_buildParams](#)

Build parameter panel on the main figure

 [ltpda_constructor_helper_class](#)

Callback executed when user selects a class

 [ltpda_constructor_helper_sets](#)

Callback executed when user selects a parameter set

Subsequent directories:

 [CVS](#)

Index for m/gui/gltpda

Matlab files in this directory:

 Contents	%% path: m/gui/gltpda %%
 buildLibrary	Function to update the library with all the methods functions.
 g_armult	This is the automatic function wrapper
 g_cleaninput	This is the automatic function wrapper
 g_copyparams	This is the automatic function wrapper
 g_iplot	This is the automatic function wrapper
 g_pl2ao	For the block plist2AO : let the user to build a plist and convert it
 g_plot	This is the automatic function wrapper
 g_resp	\$Id: g_resp.m,v 1.1 2008/03/01 13:43:20 nicola Exp \$
 g_saveresult	This is the automatic function wrapper
 loginLTPDA	LOGINDLG Dialog for visually secure login.
 ltpdagui	=====
 ltpdalib	LTPDALIB Open LTPDA Package Library.
 ltpdasim	This is the automatic function wrapper
 pan1main	This is the function for the main panel of the GUI LTPDA, the package of
 pan2param	This is the GUI m-file able to read the list of parameters to be set for
 pan4import	=====
 pan5output	=====
 pan6globals	=====
 pan7partial	=====
 panXyettocome	=====
 progressBar	Progress bar for the LTPDA GUI.
 slblocks	SLBLOCKS Defines the block library for a specific Toolbox or Blockset.
 uipickfiles	uipickfiles: GUI program to select file(s) and/or directories.
 welcome2LTPDA	Introductory window for the LTPDA Toolbox.

Other Matlab-specific files in this directory:

-  ltpdaSimConfigSet.mat
-  libraryBase.mdl
-  ltpda_library.mdl

Subsequent directories:

-  CVS

Index for m/gui/ltpdaRepoGUI

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/gui/ltpdaRepoGUI
%%%%%%%%%%%%%
-  [createTable](#) createTable - create nice-looking table based on javax.swing.JTable
-  [uitable](#) UITABLE creates a two dimensional graphic uitable component in a figure window.
-  [repogui](#) REPOGUI a GUI to interact with an LTPDA Repository.
-  [sqlResultsGUI](#) SQLRESULTSGUI shows the results of a query in a new window.
-  [uitabpanel](#) UITABPANEL adds a group of tabbed uipanel container objects to a figure.

Subsequent directories:

-  [CVS](#)
-  [callbacks](#)

Index for m/gui/ltpdaRepoGUI/callbacks

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/gui/ltpdaRepoGUI/callbacks %%%%%%%%%%%%%%
 buildConditions	Callback to build the query conditions panel.
 buildquery	Build a query from the settings on the query panel
 cb_classMethodList	Get a list of methods for a class and fill the list
 cb_condPlusBtn	Add a condition to the query panel
 cb_condSubtractBtn	Remove a condition from the query panel
 cb_connectBtn	Callback to connect to server
 cb_dbSelect	Callback for when user selects a database
 cb_executeQuery	Execute the query built on the query panel
 cb_fieldSelect	Callback executed when a field is selected
 cb_getDBs	Callback executed when the user clicks the 'Get DBs' button
 cb_importBtn	Callback executed when the user clicks on the import button.
 cb_mainfigClose	Callback executed when the main figure closes
 cb_refreshObjsList	Callback to refresh the list of objects.
 cb_refreshObjsListQL	Callback to refresh the list of objects.
 cb_retrieveBtn	Callback executed when the user clicks on the retrieve button.
 cb_saveBtn	Callback executed when the user clicks on the save button.
 cb_serverSelect	Callback executed when the user selects a server.
 cb_submitBtn	Callback executed when the user clicks on the 'Submit' button
 cb_tableSelect	Callback executed when a table selection is made
 cb_workspaceObjsList	Callback to fill the list of workspace objects
 cb_workspaceObjsListQL	Callback to fill the list of workspace objects
 fillClassMethodsList	Callback to fill the list of class methods
 getFields	Get a list of database fields
 getTables	Get a list of database tables
 getWorkspaceObjs	Get a list of LTPDA objects in the MATLAB workspace
 setWorkspaceObjsList	Fill the workspace object list
 setWorkspaceObjsListQL	Fill the workspace object list

Subsequent directories:

 [CVS](#)

Index for m/gui/ltpdalauncher

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/gui/ltpdalauncher %%%%%%%%%%%%%%
 ltpda_launcher_close	Callback executed when the main figure closes
 ltpda_launcher_launch_ltpdagui	Callback executed when the LTPDA GUI button is pressed
 ltpda_launcher_launch_pzmodel	Callback executed when the LTPDA REPO GUI button is pressed
 ltpda_launcher_launch_repogui	Callback executed when the LTPDA REPO GUI button is pressed
 ltpdalauncher	LTPDALAUNCHER presents the user with a control panel from which they can

Subsequent directories:

-  [CVS](#)
-  [images](#)

Index for m/gui/ltpdv

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/gui/ltpdv
%%%%%%%%%%%%%
-  [ltpdv](#) LTPDV dataviewer based on AOs and functions in LTPDA.
-  [make_ltpda_objects](#)
-  [scratch](#) SCRATCH M-file for scratch.fig
-  [uitabpanel](#) UITABPANEL adds a group of tabbed uipanel container objects to a figure.

Subsequent directories:

-  [CVS](#)
-  [callbacks](#)

Index for m/gui/ltpdv/callbacks

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/gui/ltpdv/callbacks %%%%%%%%%%%%%%
 ltpdv_buildFilterPanel	
 ltpdv_buildImportPanel	
 ltpdv_buildMathPanel	
 ltpdv_buildPlotPanel	
 ltpdv_buildPreprocessPanel	
 ltpdv_buildPropertiesPanel	
 ltpdv_buildSpectralPanel	
 ltpdv_build_buildobject	LTPDV_BUILD_BUILDOBJECT callback for the build object button on the build
 ltpdv_build_class_select	LTPDV_BUILD_CLASS_SELECT callback when the user selects the class
 ltpdv_build_get_selected_class	LTPDV_BUILD_GET_SELECTED_CLASS returns the currently selected class on
 ltpdv_build_get_selected_pset	LTPDV_BUILD_GET_SELECTED_PSET returns the currently selected set on
 ltpdv_build_params_panel	
 ltpdv_build_set_select	LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to
 ltpdv_calc_btn	LTPDV_CALC_BTN called when one of the calculator buttons is pushed.
 ltpdv_cb_mainfigClose	Callback executed when the main figure closes
 ltpdv_delete_objects	LTPDV_DELETE_OBJECTS delete the selected objects from the object list
 ltpdv_explore_objects	Handles
 ltpdv_export_objects	Handles
 ltpdv_fcn_run	LTPDV_FCN_RUN runs the command on the selected panel
 ltpdv_filterAdd	Handles
 ltpdv_filterApply	Handles
 ltpdv_filterBrowse	Handles
 ltpdv_filterDelete	Handles
 ltpdv_filterImport	Handles
 ltpdv_filter_control_chans	DV_FILTERCONTROLCHANS filter out control channels
 ltpdv_filter_from_standard_panel	LTPDV_FILTER_FROM_STANDARD_PANEL builds a filter from the description on
 ltpdv_filter_get_pzmodel	Get poles
 ltpdv_filter_getf	get f1
 ltpdv_filter_list_select	Handles
 ltpdv_filtersClear	Handles

 ltpdv_getWorkspaceObjs	LTPDV_GETWORKSPACEOBSJS gets a list of LTPDA user objects currently in the
 ltpdv_get_params_from_panel	LTPDV_GET_PARAMS_FROM_PANEL get a plist from the parameters on a
 ltpdv_get_selected_filter	Get selected panel
 ltpdv_get_selected_objs	LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.
 ltpdv_hist_plot	Handles
 ltpdv_importObjs	LTPDV_IMPORTOBSJS import given objects to the object list
 ltpdv_load_objects	Handles
 ltpdv_objectList_edit	LTPDV_OBJECTLIST_EDIT activates when the user changes the content of a
 ltpdv_objectList_select	LTPDV_OBJECTLIST_SELECT activates when the user selects cells in the
 ltpdv_parse_control_chans	DV_PARSECONTROLCHANS parse out a list of control channels from the input
 ltpdv_plot	Handles
 ltpdv_plotFilterResp	LTPDV_PLOTFILTERRESP plots the response of the selected filter.
 ltpdv_poleAdd	Handles
 ltpdv_poleClear	Handles
 ltpdv_poleDelete	Handles
 ltpdv_preproc_fcncist	Handles
 ltpdv_preproc_param_act	Handles
 ltpdv_preproc_sets_callback	
 ltpdv_properties_apply	LTPDV_PROPERTIES_APPLY callback for apply button on the object properties
 ltpdv_properties_build_props	Object class
 ltpdv_pzmResp	Handles
 ltpdv_pzmStore	Handles
 ltpdv_pzmStoreResp	Handles
 ltpdv_refresh_object_list	LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.
 ltpdv_save_objects	Handles
 ltpdv_search_channels	LTPDV_SEARCH_CHANNELS search the channel list and reduce it to those that
 ltpdv_server_addTime	Handles
 ltpdv_server_channellist	Handles
 ltpdv_server_clearTimes	Handles
 ltpdv_server_datatype_select	Handles
 ltpdv_server_deleteTimes	Handles
 ltpdv_server_getChannels	
 ltpdv_server_getLatest	Handles
 ltpdv_server_get_channels	Get handle to list
 ltpdv_server_get_comment	Get handle
 ltpdv_server_get_data	Get times

 ltpdv_server_get_latest_chans	Get server details
 ltpdv_server_get_server	Get data type
 ltpdv_server_get_start_time	Handle
 ltpdv_server_get_stop_time	Handle
 ltpdv_server_get_times	Do we have a list?
 ltpdv_server_import	Handles
 ltpdv_server_queryServer	Handles
 ltpdv_server_server_select	Handles
 ltpdv_server_set_channel_list	Get normal channels
 ltpdv_server_set_duration	Get handle
 ltpdv_server_set_timeslist	Get times list
 ltpdv_server_start_edit	Handles
 ltpdv_server_stop_edit	Handles
 ltpdv_server_timeslist	
 ltpdv_set_filter_list	Get filters
 ltpdv_set_pole_list	Get poles
 ltpdv_set_zero_list	Get poles
 ltpdv_spectral_fcnc	Handles
 ltpdv_storeFilter	LTPDV_STOREFILTER stores the current filter as an object in the object
 ltpdv_storeFilterResp	LTPDV_STOREFILTERRESP stores the response on the current filter in the
 ltpdv_tabChanged	
 ltpdv_workspaceList_select	LTPDV_WORKSPACELIST_SELECT activates when the user selects cells in the
 ltpdv_workspaceList_set_data	LTPDV_WORKSPACELIST_SET_DATA sets the data into the workspace table
 ltpdv_zeroAdd	Handles
 ltpdv_zeroClear	Handles
 ltpdv_zeroDelete	Handles

Subsequent directories:

 [CVS](#)

Index for m/gui/pzmodel_designer

Matlab files in this directory:

-  [Contents](#)
- %%%%%%%%%%%%%% path: m/gui/pzmodel_designer
%%%%%%%%%%%%%%
-  [pzmodel_helper](#)
- PZMODEL_HELPER M-file for pzmodel_helper.fig

Subsequent directories:

-  [CVS](#)

Index for m/gui/quicklook

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/gui/quicklook
%%%%%%%%%%%%%
-  [ltpdaquicklook](#) LTPDAQICKLOOK allows the user to quicklook LTPDA objects.

Subsequent directories:

-  [CVS](#)

Index for m/gui/specwin_viewer

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/gui/specwin_viewer %%%%%%%%%%%%%%
 ltpda_specwin_viewer_build_window	Function to build the window and display it.
 ltpda_specwin_viewer_close	Callback executed when the GUI is closed
 ltpda_specwin_viewer_wintype	Callback executed when the user selects a window
 specwin_viewer	SPECWIN_VIEWER allows the user to explore spectral windows.

Subsequent directories:

 [CVS](#)

Index for m/helper

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/helper %%%%%%%%%%%%%%%
-  [mc](#) A function to properly clear MATLAB memory for LTPDA.

Subsequent directories:

-  [CVS](#)

Index for m/math

Matlab files in this directory:

-  [Contents](#) %%%%%%%%% path: m/math
%%%%%%%%%
-  [ltpda_lincom](#) LTPDA_LINCOM make the linear combination of the input analysis objects.

Subsequent directories:

-  [CVS](#)

Constructor examples of the POLE class

[Construct empty POLE object](#)

[Construct a POLE object by loading the object from a file](#)

[Construct a POLE object with a real pole](#)

[Construct a POLE object with a complex pole](#)

Poles are specified by in the LTPDA Toolbox by a frequency, f , and (optionally) a quality factor, Q .

Construct empty POLE object

The following example creates an empty pole object

```
p = pole()
---- pole 1 ----
None: NaN Hz
-----
```

Construct a POLE object by loading the object from a file

The following example creates a new pole object by loading the pole object from disk.

```
p = pole('pole.mat')
p = pole('pole.xml')
```

Construct a POLE object with a real pole

The following code fragment creates a real pole at 1Hz:

```
p = pole(1)
---- pole 1 ----
real pole: 1 Hz
-----
```

It is also possible to use a parameter list (PLIST) object

```
pl = plist('f', 1);
p = pole(pl)
---- pole 1 ----
real pole: 1 Hz
-----
```

Construct a POLE object with a complex pole

To create a complex pole, you can specify a quality factor. For example

```
p = pole(10,4)
---- pole 1 ----
complex pole: 10 Hz, Q=4 [-0.0019894+0.015791i]
-----
```

It is also possible to use a parameter list (PLIST) object

```
pl = plist('f', 10, 'q', 4);
pole(pl)
---- pole 1 ----
complex pole: 10 Hz, Q=4 [-0.0019894+0.015791i]
-----
```

Constructor examples of the ZERO class

[Construct empty ZERO object](#)

[Construct a ZERO object by loading the object from a file](#)

[Construct a ZERO object with a real zero](#)

[Construct a ZERO object with a complex zero](#)

Poles are specified by in the LTPDA Toolbox by a frequency, f , and (optionally) a quality factor, Q .

Construct empty ZERO object

The following example creates an empty zero object

```
p = zero()
---- zero 1 ----
None: NaN Hz
-----
```

Construct a ZERO object by loading the object from a file

The following example creates a new zero object by loading the zero object from disk.

```
p = zero('zero.mat')
p = zero('zero.xml')
```

Construct a ZERO object with a real zero

The following code fragment creates a real zero at 1Hz:

```
p = zero(1)
---- zero 1 ----
real zero: 1 Hz
-----
```

It is also possible to use a parameter list (PLIST) object

```
pl = plist('f', 1);
p = zero(pl)
---- zero 1 ----
real zero: 1 Hz
-----
```

Construct a ZERO object with a complex zero

To create a complex zero, you can specify a quality factor. For example

```
p = zero(10,4)
---- zero 1 ----
complex zero: 10 Hz, Q=4 [-0.0019894+0.015791i]
-----
```

It is also possible to use a parameter list (PLIST) object

```
pl = plist('f', 10, 'q', 4);
zero(pl)
---- zero 1 ----
complex zero: 10 Hz, Q=4 [-0.0019894+0.015791i]
-----
```

Matlab Index

Matlab Directories

- 📁 [classes/@ao](#)
- 📁 [classes/@cdata](#)
- 📁 [classes/@data2D](#)
- 📁 [classes/@data3D](#)
- 📁 [classes/@fsdata](#)
- 📁 [classes/@history](#)
- 📁 [classes/@ltpda_data](#)
- 📁 [classes/@ltpda_filter](#)
- 📁 [classes/@ltpda_nuo](#)
- 📁 [classes/@ltpda_obj](#)
- 📁 [classes/@ltpda_uo](#)
- 📁 [classes/@ltpda_uoh](#)
- 📁 [classes/@mfir](#)
- 📁 [classes/@miir](#)
- 📁 [classes/@minfo](#)
- 📁 [classes/@param](#)
- 📁 [classes/@plist](#)
- 📁 [classes/@provenance](#)
- 📁 [classes/@pz](#)
- 📁 [classes/@pzmodel](#)
- 📁 [classes/@specwin](#)
- 📁 [classes/@ssm](#)
- 📁 [classes/@time](#)
- 📁 [classes/@timespan](#)
- 📁 [classes/@tsdata](#)
- 📁 [classes/@unit](#)
- 📁 [classes/@xydata](#)
- 📁 [classes/@xyzdata](#)
- 📁 [m/etc](#)
- 📁 [m/gui/@jcontrol](#)
- 📁 [m/gui/@jcontrol/private](#)
- 📁 [m/gui/ao_browser](#)
- 📁 [m/gui/constructor_helper](#)
- 📁 [m/gui/gltpda](#)
- 📁 [m/gui/ltpdaRepoGUI](#)
- 📁 [m/gui/ltpdaRepoGUI/callbacks](#)
- 📁 [m/gui/ltpdalauncher](#)
- 📁 [m/gui/ltpdv](#)
- 📁 [m/gui/ltpdv/callbacks](#)
- 📁 [m/gui/pzmodel_designer](#)
- 📁 [m/gui/quicklook](#)
- 📁 [m/gui/specwin_viewer](#)
- 📁 [m/helper](#)
- 📁 [m/math](#)
- 📁 [m/sigproc/frequency_domain](#)
- 📁 [m/sigproc/time_domain](#)
- 📁 [m/ssm_models](#)

Matlab Files found in these Directories

Contents	dropduplicates	ltpdv_fcn_run	setEnbw
Contents	dsmean	ltpdv_filterAdd	setEndT
Contents	eig	ltpdv_filterApply	setEpochtime
Contents	epoch2str	ltpdv_filterBrowse	setF
Contents	eq	ltpdv_filterDelete	setFs
Contents	exp	ltpdv_filterImport	setFs
Contents	export	ltpdv_filter_control_chans	setFs
Contents	extractm	ltpdv_filter_from_standard_panel	setFs
Contents	extractmdl	ltpdv_filter_get_pzmodel	setGain
Contents	fft	ltpdv_filter_getf	setGd
Contents	fillClassMethodsList	ltpdv_filter_list_select	setHist

Contents	filload	ltpdv_filtersClear	setHistin
Contents	filter	ltpdv_getWorkspaceObjs	setHistout
Contents	filtfilt	ltpdv_get_params_from_panel	setInfile
Contents	find	ltpdv_get_selected_filter	setKey
Contents	find	ltpdv_get_selected_objjs	setKeyVal
Contents	findFsMax	ltpdv_hist_plot	setMversion
Contents	findShortestVector	ltpdv_importObjs	setName
Contents	firwhiten	ltpdv_load_objects	setName
Contents	fitfs	ltpdv_objectList_edit	setNavs
Contents	fixNsecs	ltpdv_objectList_select	setNsecs
Contents	fixfs	ltpdv_parse_control_chans	setPoles
Contents	fngen	ltpdv_plot	setQ
Contents	fngen	ltpdv_plotFilterResp	setRI
Contents	format	ltpdv_poleAdd	setStartT
Contents	fq2fac	ltpdv_poleClear	setT0
Contents	fq2ri	ltpdv_poleDelete	setT0
Contents	fromA	ltpdv_preproc_fcncList	setT0
Contents	fromAB	ltpdv_preproc_param_act	setTime_str
Contents	fromAO	ltpdv_preproc_sets_callback	setTimeformat
Contents	fromDataInMAT	ltpdv_properties_apply	setTimeformat
Contents	fromDataInMAT	ltpdv_properties_build_props	setTimezone
Contents	fromDatafile	ltpdv_pzmResp	setTimezone
Contents	fromDatafile	ltpdv_pzmStore	setVal
Contents	fromFSfcn	ltpdv_pzmStoreResp	setVersion
Contents	fromFcn	ltpdv_refresh_object_list	setWorkspaceObjsList
Contents	fromFile	ltpdv_save_objects	setWorkspaceObjsListQL
Contents	fromFile	ltpdv_search_channels	setX
Contents	fromLISO	ltpdv_server_addTime	setX
Contents	fromLISO	ltpdv_server_channellist	setXY
Contents	fromLISO	ltpdv_server_clearTimes	setXY
Contents	fromPolesAndZeros	ltpdv_server_datatype_select	setXunits
Contents	fromPolyval	ltpdv_server_deleteTimes	setXunits
Contents	fromPzmmodel	ltpdv_server_getChannels	setY
Contents	fromPzmmodel	ltpdv_server_getLatest	setY
Contents	fromPzmmodel	ltpdv_server_get_channels	setY
Contents	fromRepository	ltpdv_server_get_comment	setYunits
VisibleProperty	fromRepository	ltpdv_server_get_data	setYunits
abccascade	fromSpecWin	ltpdv_server_get_latest_chans	setYunits
abs	fromStandard	ltpdv_server_get_server	setZ
acos	fromStandard	ltpdv_server_get_start_time	setZ
addHistory	fromTSfcn	ltpdv_server_get_stop_time	setZeros
ancestor	fromTimeString	ltpdv_server_get_times	setZunits
ao	fromTimeValue	ltpdv_server_import	setappdata
ao2m	fromTimespanDef	ltpdv_server_queryServer	sign
ao_fcn_template	fromVals	ltpdv_server_server_select	simulate
append	fromWaveform	ltpdv_server_set_channel_list	sin
applymethod	fs	ltpdv_server_set_duration	slblocks
applymethod	fsdata	ltpdv_server_set_timeslist	smoother
applymethod	g_arrmult	ltpdv_server_start_edit	sort
applyoperator	g_cleaninput	ltpdv_server_stop_edit	spectrogram
applyoperator	g_copyparams	ltpdv_server_timeslist	specwin
applyoperator	g_iplot	ltpdv_set_filter_list	specwin_viewer
asin	g_pl2ao	ltpdv_set_pole_list	spikecleaning
assemble	g_plot	ltpdv_set_zero_list	split
atan	g_resp	ltpdv_spectral_fcncList	sqlResultsGUI
atan2	g_saveresult	ltpdv_storeFilter	sqrt
attachm	gapfilling	ltpdv_storeFilterResp	sqrt
attachmdl	ge	ltpdv_tabChanged	ssm
buildConditions	get	ltpdv_workspaceList_select	ssm2iirpz

buildLibrary	getFfromYFs	ltpdv_workspaceList_set_data	ssm2ss
buildquery	getFields	ltpdv_zeroAdd	ssmFromBuiltinSystem
cat	getNodes	ltpdv_zeroClear	ssmFromDescription
cb_classMethodList	getNodes_plot	ltpdv_zeroDelete	ssmFromFilename
cb_condPlusBtn	getTables	lxspec	ssmFromMiir
cb_condSubtractBtn	getTimezones	make_ltpda_objects	ssmFromPzmodel
cb_connectBtn	getWorkspaceObjs	max	ssmFromRepository
cb_dbSelect	getX	mc	ssmFromss
cb_executeQuery	getX	mchol	ssm_model_delay
cb_fieldSelect	getY	md5	ssm_model_standard_system_noparams
cb_getDBs	getY	mdc1_ifo2acc_fd	ssm_model_standard_system_params
cb_importBtn	getZ	mdc1_ifo2acc_inloop	std
cb_mainfigClose	get_window	mdc1_ifo2control	string
cb_refreshObjsList	getappdata	mdc1_input_noises	string
cb_refreshObjsListQL	getlowerFreq	mdc1_x2acc	string
cb_retrieveBtn	getupperFreq	mean	string
cb_saveBtn	growT	median	string
cb_serverSelect	gt	method_fcn_template	string
cb_submitBtn	hist	mfir	string
cb_tableSelect	hist2dot	miir	string
cb_workspaceObjsList	hist2m	min	string
cb_workspaceObjsListQL	history	minfo	string
cdata	ifft	minreal	string
cell_add	imag	minus	string
cell_fusion	index	minus	string
cell_mult	interp	mkbandpass	string
cell_recut	interpmissing	mkbandpass	submit
cell_select	inv	mkbandreject	subsasgn
cellstrfind	iplot	mkbandreject	subsref
char	isappdata	mkhighpass	sum
char	isparam	mkhighpass	svd
char	isprop	mklowpass	t0
char	isstable	mklowpass	tan
char	jcontrol	mlpsd_m	tfe
char	join	mlpsd_mex	time
char	kaiser_alpha	mltfe	timedomainfit
char	kaiser_flatness	mode	times
char	kaiser_nenbw	modifparams	times
char	kaiser_rov	modiftimestep	timeshift
char	kaiser_w3db	modify	timespan
char	kalman	mpower	tomfir
char	lcohere	mpower	tomiir
char	lcpd	mrdivide	transpose
char	le	mrdivide	tsdata
char	len	mtimes	type
char	lincom	mtimes	uipickfiles
char	linedetect	muitable	uitabpanel
close	ln	mux	uitabpanel
cohere	log	ne	uminus
collapseX	log10	ngconv	unit
combine	loginLTPDA	nginit	update_struct
complex	lpsd	ngprop	update_struct
compute	lt	ngsetup	update_struct
computeDFT	ltf_plan	ngsetup_vpa	update_struct
computeInterval	ltfe	norm	update_struct
computeperiodogram	ltpda_cohere	nparams	update_struct
conj	ltpda_constructor_helper	pan1main	update_struct
consolidate	ltpda_constructor_helper_build	pan2param	update_struct
conv	ltpda_constructor_helper_buildParams	pan4import	update_struct

conv_noisegen	ltpda_constructor_helper_class	pan5output	update_struct
copy	ltpda_constructor_helper_sets	pan6globals	update_struct
copy	ltpda_cpsd	pan7partial	update_struct
copy	ltpda_data	panXyettocome	update_struct
copy	ltpda_explorer	param	update_struct
copy	ltpda_filter	parseFilterParams	update_struct
copy	ltpda_gapfilling	parseFilterParams	update_struct
copy	ltpda_javacomponent	parse_time_string	update_struct
copy	ltpda_launcher_close	phase	update_struct
copy	ltpda_launcher_launch_ltpdagui	phasetrack	update_struct
copy	ltpda_launcher_launch_pzmodel	plist	update_struct
copy	ltpda_launcher_launch_repogui	plot	upsample
copy	ltpda_lcohere	plot	validate
copy	ltpda_lcpsd	plot	validate
copy	ltpda_lincom	plus	var
copy	ltpda_linedetect	plus	welch
copy	ltpda_lpsd	plus	welchparse
copy	ltpda_ltf	polyfit	welchscale
copy	ltpda_nfest	power	welcome2LTPDA
copy	ltpda_nuo	power	win_BH92
copy	ltpda_obj	prependVersion	win_Bartlett
cos	ltpda_polydetrend	progressBar	win_FTHP
cp2iir	ltpda_pwelch	provenance	win_FTNI
cpsd	ltpda_rms	psd	win_FTSRS
createTable	ltpda_specwin_viewer_build_window	pset	win_HFT116D
ctranspose	ltpda_specwin_viewer_close	pwelch	win_HFT144D
cz2iir	ltpda_specwin_viewer_wintype	pz	win_HFT169D
data2D	ltpda_spikecleaning	pzm2ab	win_HFT196D
data3D	ltpda_startup	pzmodel	win_HFT223D
delay	ltpda_tfe	pzmodel_helper	win_HFT248D
delete	ltpda_uitree	rdivide	win_HFT70
demux	ltpda_uitreenode	rdivide	win_HFT90D
det	ltpda_uo	real	win_HFT95
detrend	ltpda_uoh	redesign	win_Hamming
dft	ltpda_user_fcn_template	redesign	win_Hanning
diag	ltpda_versions	reduce	win_Nuttall3
diff	ltpda_xcorr	reduce_model	win_Nuttall3a
display	ltpdagui	remove	win_Nuttall3b
display	ltpdalauncher	repogui	win_Nuttall4
display	ltpdalib	resample	win_Nuttall4a
display	ltpdaquicklook	resp	win_Nuttall4b
display	ltpdasim	resp	win_Nuttall4c
display	ltpdv	resp	win_Rectangular
display	ltpdv_buildFilterPanel	resp	win_SFT3F
display	ltpdv_buildImportPanel	resp	win_SFT3M
display	ltpdv_buildMathPanel	retrieve	win_SFT4F
display	ltpdv_buildPlotPanel	ri2fq	win_SFT4M
display	ltpdv_buildPreprocessPanel	rmappdata	win_SFT5F
display	ltpdv_buildPropertiesPanel	rms	win_SFT5M
display	ltpdv_buildSpectralPanel	rp2iir	win_Welch
display	ltpdv_build_buildobject	rz2iir	x
display	ltpdv_build_class_select	save	xcorr
display	ltpdv_build_get_selected_class	scratch	xspect
display	ltpdv_build_get_selected_pset	search	xunits
display	ltpdv_build_params_panel	select	xydata
display	ltpdv_build_set_select	set	xyzdata
display	ltpdv_calc_btn	setA	y
display2	ltpdv_cb_mainfigClose	setB	yunits
dopplercorr	ltpdv_delete_objects	setCreated	zeropad

double	ltpdv_explore_objects	setCreator
downsample	ltpdv_export_objects	setDescription

Generated on Mon 08-Sep-2008 13:18:25 by [m2html](#) © 2003

applymethod

PURPOSE ^

APPLYMETHOD to the analysis object

SYNOPSIS ^

function applymethod(as, ao_invars, method, pl, dpl, info)

DESCRIPTION ^

```
APPLYMETHOD to the analysis object
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: APPLYMETHOD to the analysis object
             Private AO function that applies the given method to
             the given AOs. This is called by all the simple methods like
             abs, mean, acos, etc.

CALL:       as = applymethod(as, ao_invars, method, pl-input, pl-default, info)

VERSION:    $Id: applymethod.m,v 1.9 2008/09/02 13:12:57 ingo Exp $

HISTORY:    11-06-2008 Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD to the analysis object

This function is called by:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [acos](#) ACOS overloads the acos method for Analysis objects.
-  [ao](#) AO analysis object class constructor.
-  [applymethod](#) APPLYMETHOD to the analysis object
-  [asin](#) ASIN overloads the asin method for Analysis objects.
-  [atan](#) ATAN overloads the atan method for Analysis objects.
-  [conj](#) CONJ overloads the conjugate operator for Analysis objects.
-  [cos](#) COS overloads the cos operator for Analysis objects. Cosine of argument in radians.
-  [det](#) DET overloads the determinant function for Analysis objects.
-  [diag](#) DIAG overloads the diagonal operator for Analysis Objects.
-  [eig](#) EIG overloads the determinant function for Analysis objects.
-  [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
-  [imag](#) IMAG overloads the imaginary operator for Analysis objects.
-  [inv](#) INV overloads the inverse function for Analysis Objects.
-  [ln](#) LN overloads the log operator for Analysis objects. Natural logarithm.
-  [log](#) LOG overloads the log operator for Analysis objects. Natural logarithm.
-  [log10](#) LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.
-  [mean](#) MEAN computes the mean value of the data in the AO.
-  [median](#) MEDIAN computes the median value of the data in the AO.

-  **mode** MODE computes the modal value of the data in the AO.
-  **norm** NORM overloads the norm operator for Analysis Objects.
-  **phase** PHASE overloads the ltpda_phase operator for Analysis objects.
-  **real** REAL overloads the real operator for Analysis objects.
-  **sign** SIGN overloads the sign operator for Analysis objects.%
-  **sin** SIN overloads the sin method for Analysis objects.
-  **sqrt** SQRT computes the square root of the data in the AO.
-  **std** STD computes the standard deviation of the data in the AO.
-  **sum** SUM computes the sum of the data in the AO.
-  **svd** SVD overloads the determinant function for Analysis objects.
-  **tan** TAN overloads the tan method for Analysis objects.
-  **uminus** UMINUS overloads the uminus operator for Analysis objects.
-  **var** VAR computes the variance of the data in the AO.

SOURCE CODE ^

```

0001 % APPLYMETHOD to the analysis object
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: APPLYMETHOD to the analysis object
0005 %               Private AO function that applies the given method to
0006 %               the given AOs. This is called by all the simple methods like
0007 %               abs, mean, acos, etc.
0008 %
0009 % CALL:         as = applymethod(as, ao_invars, method, pl-input, pl-default, info)
0010 %
0011 % VERSION:      $Id: applymethod.m,v 1.9 2008/09/02 13:12:57 ingo Exp $
0012 %
0013 % HISTORY:      11-06-2008 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function applymethod(as, ao_invars, method, pl, dpl, info)
0019
0020     import utils.const.*
0021
0022     for j=1:numel(as)
0023         % Message
0024         utils.helper.msg(msg.PROC1, 'applying %s to %s ', method, as(j).name);
0025         % Apply method to data
0026         applymethod(as(j).data, pl, plist('method', method), dpl);
0027         %combine the input plist and the default plist for the history
0028         plh = combine(pl, dpl);
0029         % append history
0030         as(j).addHistory(info, plh, ao_invars(j), as(j).hist);
0031         % Set new AO name
0032         as(j).setName([method '(' ao_invars{j} ')'], 'internal');
0033     end
0034
0035 end
0036
0037

```

applyoperator

PURPOSE ^

APPLYOPERATOR to the analysis object

SYNOPSIS ^

function res = applyoperator(as, ao_invars, op, opsym, pl, info)

DESCRIPTION ^

```
APPLYOPERATOR to the analysis object
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: APPLYOPERATOR to the analysis object
             Private AO function that applies the given operator to
             the given AOs. This is called by all the simple methods like
             plus, minus, mtimes etc.

CALL:       as = applyoperator(as, ao_invars, op, opsym, pl, info)

VERSION:    $Id: applyoperator.m,v 1.18 2008/09/05 14:13:00 hewitson Exp $

HISTORY:    11-06-2008 Hewitson
             Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applyoperator](#) APPLYOPERATOR to the analysis object

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [applyoperator](#) APPLYOPERATOR to the analysis object
-  [atan2](#) ATAN2 overloads the atan2 operator for Analysis objects. Four quadrant inverse tangent.
-  [minus](#) MINUS implements minus operator for analysis objects.
-  [mpower](#) MPOWER implements mpower operator for analysis objects.
-  [mrdivide](#) MRDIVIDE implements mrdivide operator for analysis objects.
-  [mtimes](#) MTIMES implements mtimes operator for analysis objects.
-  [plus](#) PLUS implements addition operator for analysis objects.
-  [power](#) POWER implements power operator for analysis objects.
-  [rdivide](#) RDIVIDE implements rdivide operator for analysis objects.
-  [times](#) TIMES implements times operator for analysis objects.

SOURCE CODE ^

```
0001 % APPLYOPERATOR to the analysis object
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: APPLYOPERATOR to the analysis object
0005 %             Private AO function that applies the given operator to
0006 %             the given AOs. This is called by all the simple methods like
0007 %             plus, minus, mtimes etc.
0008 %
```

```

0009 % CALL:      as = applyoperator(as, ao_invars, op, opsym, pl, info)
0010 %
0011 % VERSION:    $Id: applyoperator.m,v 1.18 2008/09/05 14:13:00 hewitson Exp $
0012 %
0013 % HISTORY:    11-06-2008 Hewitson
0014 %              Creation
0015 %
0016 %%%%%%%%%%%
0017
0018 function res = applyoperator(as, ao_invars, op, opsym, pl, info)
0019
0020 %% Initialise the result to the first input object
0021 res = as(1);
0022
0023 %% go through the remaining analysis objects
0024 for j=2:numel(as)
0025
0026     % Message
0027     utils.helper.msg(2, 'applying %s to %s and %s', op, res.name, as(j).name);
0028
0029     % Compute operator of data
0030     yu1 = res.data.yunits;
0031     yu2 = as(j).data.yunits;
0032     res.data = applyoperator(res.data, as(j).data, pl, plist('op', op));
0033
0034     % append history
0035     res.addHistory(info, pl, [{res.name} ao_invars(j)], [res.hist as(j).hist]);
0036
0037     % Set new AO name
0038     if (length(opsym) == 2 && opsym(1) == '.') || ...
0039        (length(opsym) == 1)
0040         res.setName(['(' res.name ') ' opsym '(' ao_invars{j} ')'], 'internal');
0041     else
0042         res.setName([opsym '(' res.name ', ' ao_invars{j} ')'], 'internal');
0043     end
0044
0045     if any(strcmp(op, {'plus', 'minus'}))
0046         if isempty(res.data.yunits.strs)
0047             res.setYunits(as(j).data.yunits);
0048         elseif isempty(as(j).data.yunits.strs)
0049             % leave the units alone
0050         elseif res.data.yunits ~= as(j).data.yunits
0051             error(['### You can not apply ' opsym ' to data with different Y units']);
0052         end
0053     elseif ismethod('unit', op)
0054         % Set units
0055         if any(strcmp(op, {'mpower', 'power'}))
0056             res.setYunits(feval(op, yu1, as(j).data.getY), 'internal');
0057         else
0058             res.setYunits(feval(op, yu1, yu2), 'internal');
0059         end
0060     else
0061         warning('LTPDA:INFO', '### This method doesn't exist in the units class. Please
0062         set the units yourself. ');
0063     end
0064 end
0065
0066 end
0067

```

[Home](#) > [classes](#) > [@ao](#) > computeDFT.m

computeDFT

PURPOSE [^]

COMPUTEDFT Computes DFT using FFT or Goertzel

SYNOPSIS [^]

function [Xx,f] = computeDFT(xin,nfft,varargin)

DESCRIPTION [^]

```

COMPUTEDFT Computes DFT using FFT or Goertzel
This function is used to calculate the DFT of a signal using the FFT
or the Goertzel algorithm.

[XX,F] = COMPUTEDFT(XIN,NFFT) where NFFT is a scalar and computes the
DFT XX using FFT. F is the frequency points at which the XX is
computed and is of length NFFT.

[XX,F] = COMPUTEDFT(XIN,F) where F is a vector with atleast two
elements computes the DFT XX using the Goertzel algorithm.

[XX,F] = COMPUTEDFT(...,Fs) returns the frequency vector F (in hz)
where Fs is the sampling frequency

Inputs:
XIN is the input signal
NFFT if a scalar corresponds to the number of FFT points used to
calculate the DFT using FFT.
NFFT if a vector corresponds to the frequency points at which the DFT
is calculated using goertzel.
FS is the sampling frequency

A direct copy of MATLAB's function for LTPDA
M Hewitson 08-05-08

$Id: computeDFT.m,v 1.2 2008/08/01 13:19:42 ingo Exp $

```

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
-  [fft](#) FFT overloads the fft method for Analysis objects.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS [^]

-  [function \[Xx,f\] = computeDFTviaFFT\(xin,nx,nfft,Fs\)](#)
-  [function \[Xx,f\] = computeDFTviaGoertzel\(xin,freqvec,Fs\)](#)

SOURCE CODE [^]

```

0001 % COMPUTEDFT Computes DFT using FFT or Goertzel
0002 % This function is used to calculate the DFT of a signal using the FFT
0003 % or the Goertzel algorithm.
0004 %

```

```

0005 % [XX,F] = COMPUTEDFT(XIN,NFFT) where NFFT is a scalar and computes the
0006 % DFT XX using FFT. F is the frequency points at which the XX is
0007 % computed and is of length NFFT.
0008 %
0009 % [XX,F] = COMPUTEDFT(XIN,F) where F is a vector with atleast two
0010 % elements computes the DFT XX using the Goertzel algorithm.
0011 %
0012 % [XX,F] = COMPUTEDFT(...,Fs) returns the frequency vector F (in hz)
0013 % where Fs is the sampling frequency
0014 %
0015 % Inputs:
0016 % XIN is the input signal
0017 % NFFT if a scalar corresponds to the number of FFT points used to
0018 % calculate the DFT using FFT.
0019 % NFFT if a vector corresponds to the frequency points at which the DFT
0020 % is calculated using goertzel.
0021 % FS is the sampling frequency
0022 %
0023 % A direct copy of MATLAB's function for LTPDA
0024 %
0025 % M Hewitson 08-05-08
0026 %
0027 % $Id: computeDFT.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0028 %
0029
0030 % Copyright 2006 The MathWorks, Inc.
0031
0032 % [1] Oppenheim, A.V., and R.W. Schaffer, Discrete-Time Signal Processing,
0033 % Prentice-Hall, Englewood Cliffs, NJ, 1989, pp. 713-718.
0034 % [2] Mitra, S. K., Digital Signal Processing. A Computer-Based Approach.
0035 % 2nd Ed. McGraw-Hill, N.Y., 2001.
0036
0037 function [Xx,f] = computeDFT(xin,nfft,varargin)
0038
0039 error(nargchk(2,3,nargin,'struct'));
0040 if nargin > 2,
0041     Fs = varargin{1};
0042 else
0043     Fs = 2*pi;
0044 end
0045
0046 nx = size(xin,1);
0047
0048 if length(nfft) > 1,
0049     isfreqVector = true;
0050 else
0051     isfreqVector = false;
0052 end
0053
0054 if ~isfreqVector,
0055     [Xx,f] = computeDFTviaFFT(xin,nx,nfft,Fs);
0056 else
0057     [Xx,f] = computeDFTviaGoertzel(xin,nfft,Fs);
0058 end
0059
0060 end
0061
0062 %-----
0063 function [Xx,f] = computeDFTviaFFT(xin,nx,nfft,Fs)
0064 % Use FFT to compute raw STFT and return the F vector.
0065
0066 % Handle the case where NFFT is less than the segment length, i.e., "wrap"
0067 % the data as appropriate.
0068 xin_ncol = size(xin,2);
0069 xw = zeros(nfft,xin_ncol);
0070 if nx > nfft,
0071     for j = 1:xin_ncol,
0072         xw(:,j) = datawrap(xin(:,j),nfft);
0073     end
0074 else
0075     xw = xin;
0076 end
0077
0078 Xx = fft(xw,nfft);
0079 f = psdfreqvec('npts',nfft,'Fs',Fs);
0080 end
0081
0082 %-----
0083 function [Xx,f] = computeDFTviaGoertzel(xin,freqvec,Fs)
0084 % Use Goertzel to compute raw DFT and return the F vector.
0085
0086 f = freqvec(:);
0087 f = mod(f,Fs); % 0 <= f <= Fs
0088 nflld = floor(freqvec(:)/Fs);
0089 xm = size(xin,1); % NFFT
0090

```



```

0091 % Indices used by the Goertzel function (see equation 11.1 pg. 755 of [2])
0092 fscaled = f/Fs*xm+1;
0093 k = round(fscaled);
0094
0095 % shift for each frequency from default xm length grid
0096 deltak = fscaled-k;
0097
0098 tempk = k;
0099 % If k > xm, fold over to the 1st bin
0100 k(tempk > xm) = 1;
0101 nflld = nflld + (tempk > xm); % Make nflld reflect fold in k because of round
0102
0103 n = (0:xm-1)';
0104 Xx = zeros(size(k,1),size(xin,2));
0105 for kindex = 1:length(k)
0106     % We need to evaluate the DFT at the requested frequency instead of a
0107     % neighboring frequency that lies on the grid obtained with xm number
0108     % of points in the 0 to fs range. We do that by giving a complex phase
0109     % to xin equal to the offset from the frequency to its nearest neighbor
0110     % on the grid. This phase translates into a shift in the DFT by the
0111     % same amount. The Xx(k) then is the DFT at (k+deltak).
0112
0113     % apply kernal to xin so as to evaluate DFT at k+deltak)
0114     kernel = exp(-j*2*pi*deltak(kindex)/xm*n);
0115     xin_phaseshifted = xin.*repmat(kernel,1,size(xin,2));
0116
0117     Xx(kindex,:) = goertzel(xin_phaseshifted,k(kindex));
0118 end
0119
0120 % DFT computed at exactly the frequencies it was requested for
0121 f = freqvec(:);
0122 end
0123
0124

```

computeperiodogram

PURPOSE ^

COMPUTEPERIODOGRAM Periodogram spectral estimation.

SYNOPSIS ^

function [P,f] = computeperiodogram(x,win,nfft,esttype,varargin)

DESCRIPTION ^

```
COMPUTEPERIODOGRAM    Periodogram spectral estimation.
This function is used to calculate the Power Spectrum Sxx, and the
Cross Power Spectrum Sxy.

Sxx = COMPUTEPERIODOGRAM(X,WIN,NFFT) where x is a vector returns the
Power Spectrum over the whole Nyquist interval, [0, 2pi).

Sxy = COMPUTEPERIODOGRAM({X,Y},WIN,NFFT) returns the Cross Power
Spectrum over the whole Nyquist interval, [0, 2pi).

Inputs:
X          - Signal vector or a cell array of two elements containing
             two signal vectors.
WIN        - Window
NFFT       - Number of frequency points (FFT) or vector of
             frequencies at which periodogram is desired (Goertzel)
WINCOMPFLAG - A string indicating the type of window compensation to
             be done. The choices are:
             'ms' - compensate for Mean-square (Power) Spectrum;
                   maintain the correct power peak heights.
             'psd' - compensate for Power Spectral Denstiy (PSD);
                   maintain correct area under the PSD curve.

Output:
Sxx        - Power spectrum [Power] over the whole Nyquist interval.
or
Sxy        - Cross power spectrum [Power] over the whole Nyquist
             interval.

A direct copy of MATLAB's function for LTPDA
M Hewitson 08-05-08

$Id: computeperiodogram.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [conj](#) CONJ overloads the conjugate operator for Analysis objects.
-  [sum](#) SUM computes the sum of the data in the AO.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function \[x,Lx,y,is2sig,win,errid,errmsg\] = validateinputs\(x,win,nfft\)](#)

SOURCE CODE ^

```

0001 % COMPUTEPERIODOGRAM Periodogram spectral estimation.
0002 % This function is used to calculate the Power Spectrum Sxx, and the
0003 % Cross Power Spectrum Sxy.
0004 %
0005 % Sxx = COMPUTEPERIODOGRAM(X,WIN,NFFT) where x is a vector returns the
0006 % Power Spectrum over the whole Nyquist interval, [0, 2pi).
0007 %
0008 % Sxy = COMPUTEPERIODOGRAM({X,Y},WIN,NFFT) returns the Cross Power
0009 % Spectrum over the whole Nyquist interval, [0, 2pi).
0010 %
0011 % Inputs:
0012 % X - Signal vector or a cell array of two elements containing
0013 % two signal vectors.
0014 % WIN - Window
0015 % NFFT - Number of frequency points (FFT) or vector of
0016 % frequencies at which periodogram is desired (Goertzel)
0017 % WINCOMPFLAG - A string indicating the type of window compensation to
0018 % be done. The choices are:
0019 % 'ms' - compensate for Mean-square (Power) Spectrum;
0020 % maintain the correct power peak heights.
0021 % 'psd' - compensate for Power Spectral Density (PSD);
0022 % maintain correct area under the PSD curve.
0023 %
0024 % Output:
0025 % Sxx - Power spectrum [Power] over the whole Nyquist interval.
0026 % or
0027 % Sxy - Cross power spectrum [Power] over the whole Nyquist
0028 % interval.
0029 %
0030 % A direct copy of MATLAB's function for LTPDA
0031 %
0032 % M Hewitson 08-05-08
0033 %
0034 % $Id: computeperiodogram.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0035 %
0036 %
0037 % Author(s): P. Pacheco
0038 % Copyright 1988-2006 The MathWorks, Inc.
0039 % $Revision: 1.2 $ $Date: 2008/08/01 13:19:42 $
0040
0041 function [P,f] = computeperiodogram(x,win,nfft,esttype,varargin)
0042
0043 error(nargchk(3,5,nargin,'struct'));
0044 if nargin < 4,
0045     esttype = 'psd'; % Default, compensate for window's power.
0046 end
0047
0048 if nargin < 5 || isempty(varargin{1}),
0049     Fs = 2*pi;
0050 else
0051     Fs = varargin{1};
0052 end
0053
0054 % Validate inputs and convert row vectors to column vectors.
0055 [x,Lx,y,is2sig,win,errid,errmsg] = validateinputs(x,win,nfft);
0056 if ~isempty(errmsg), error(errid, errmsg); end
0057
0058 % Determine if FFT or Goertzel
0059 freqVectorSpecified = false;
0060 if (length(nfft)> 1), freqVectorSpecified = true; end
0061
0062 % Window the data
0063 xw = x.*win;
0064 if is2sig, yw = y.*win; end
0065
0066 % Evaluate the window normalization constant. A 1/N factor has been
0067 % omitted since it will cancel below.
0068 if strcmpi(esttype,'ms'),
0069     % The window is convolved with every power spectrum peak, therefore
0070     % compensate for the DC value squared to obtain correct peak heights.
0071     U = sum(win)^2;
0072 else
0073     U = win'*win; % compensates for the power of the window.
0074 end
0075
0076 % Compute the periodogram power spectrum [Power] estimate
0077 % A 1/N factor has been omitted since it cancels
0078
0079 [Xx,f] = ao.computedFT(xw,nfft,Fs);
0080 if is2sig, [Yy,f] = ao.computedFT(yw,nfft,Fs); end
0081
0082 P = Xx.*conj(Xx)/U; % Auto spectrum.

```

```

0083     if is2sig,
0084         P = Xx.*conj(Yy)/U; % Cross spectrum.
0085     end
0086
0087 end
0088
0089 %-----
0090 function [x,Lx,y,is2sig,win,errid,errmsg] = validateinputs(x,win,nfft)
0091 % Validate the inputs to computexperiodogram and convert the input signal
0092 % vectors into column vectors.
0093
0094 errid = '';
0095 errmsg = ''; % in case of early return.
0096
0097 % Set defaults and convert to row vectors to columns.
0098 y = [];
0099 Ly = 0;
0100 is2sig= false;
0101 win = win(:);
0102 Lw = length(win);
0103
0104 % Determine if one or two signal vectors was specified.
0105 Lx = length(x);
0106 if iscell(x),
0107     if length(x) > 1,
0108         y = x{2};
0109         y = y(:);
0110         is2sig = true;
0111     end
0112     x = x{1};
0113     Lx = length(x);
0114 end
0115 x = x(:);
0116
0117 if is2sig,
0118     Ly = length(y);
0119     if Lx ~= Ly,
0120         errid = generatemsgid('invalidInputSignalLength');
0121         errmsg = 'The length of the two input vectors must be equal to calculate the
cross spectral density.';
0122         return;
0123     end
0124 end
0125
0126 if Lx ~= Lw,
0127     errid = generatemsgid('invalidWindow');
0128     errmsg = 'The WINDOW must be a vector of the same length as the signal.';
0129     return;
0130 end
0131
0132 end
0133
0134

```

[Home](#) > [classes](#) > [@ao](#) > conv_noisegen.m

conv_noisegen

PURPOSE [^]

CONV_NOISEGEN calls the matlab function conv.m to convolute poles and zeros from a given pzmodel

SYNOPSIS [^]

```
function [b,a] = conv_noisegen(pol,zer)
```

DESCRIPTION [^]

CONV_NOISEGEN calls the matlab function conv.m to convolute poles and zeros from a given pzmodel

The function is a private one and is called from ngconv.m in the noisegenerator folder.

Inputs (from ngconv.m):

- pol
- zero

Outputs:

- b: denominator coefficients of transfer function
- a: numerator coefficients of transfer function

A Monsky 24-07-07

\$Id: conv_noisegen.m,v 1.2 2008/08/01 13:19:42 ingo Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [conv](#) CONV vector convolution.
-  [sqrt](#) SQRT computes the square root of the data in the AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE [^]

```

0001 % CONV_NOISEGEN calls the matlab function conv.m to convolute poles and zeros from a
0002 % given pzmodel
0003 % The function is a private one and is called from ngconv.m in the
0004 % noisegenerator folder.
0005 %
0006 % Inputs (from ngconv.m):
0007 %     - pol
0008 %     - zero
0009 %
0010 % Outputs:
0011 %     - b: denominator coefficients of transfer function
0012 %     - a: numerator coefficients of transfer function
0013 % A Monsky 24-07-07
0014 %
0015 % $Id: conv_noisegen.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0016 %
0017
0018 function [b,a] = conv_noisegen(pol,zer)
0019
0020     [m,k] = size(pol);
0021     [n,l] = size(zer);

```

```
0022
0023     coefb = pol(1,:);
0024
0025     for i = 2:m
0026         coefb = conv(coefb, pol(i,:));
0027     end
0028
0029     b = nonzeros(coefb);
0030
0031     if n~=0
0032         coefa = zer(1,:);
0033         for i = 2:n
0034             coefa = conv(coefa, zer(i,:));
0035         end
0036         a = nonzeros(coefa);
0037     else
0038         a = 1;
0039     end
0040
0041     %normalize to bn = 1
0042     m = length(b);
0043     normfac = b(m);
0044     b = b/normfac;
0045     a = a/(normfac*sqrt(2));
0046
0047 end
0048
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

findFsMax

PURPOSE ^

findFsMax Returns the max Fs of a set of AOs

SYNOPSIS ^

function fs = findFsMax(as)

DESCRIPTION ^

```
findFsMax Returns the max Fs of a set of AOs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: Returns the max Fs of a set of AOs
CALL:      fsmax = findFsMax(as)
VERSION:    $Id: findFsMax.m,v 1.4 2008/08/01 13:19:42 ingo Exp $
HISTORY:    14-05-07 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 **fs** FS Get the data property 'fs'.

This function is called by:

-  **ao** AO analysis object class constructor.
-  **xcorr** XCORR makes cross-correlation estimates of the time-series

SOURCE CODE ^

```
0001 % findFsMax Returns the max Fs of a set of AOs
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Returns the max Fs of a set of AOs
0005 %
0006 % CALL:      fsmax = findFsMax(as)
0007 %
0008 % VERSION:    $Id: findFsMax.m,v 1.4 2008/08/01 13:19:42 ingo Exp $
0009 %
0010 % HISTORY:    14-05-07 M Hewitson
0011 %              Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014
0015 function fs = findFsMax(as)
0016
0017     fs = 0;
0018     for j=1:length(as)
0019         a = as(j);
0020         if a.data.fs > fs
0021             fs = a.data.fs;
0022         end
0023     end
0024
0025 end
0026
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

findShortestVector

PURPOSE ^

findShortestVector Returns the length of the shortest vector in samples

SYNOPSIS ^

function lmin = findShortestVector(as)

DESCRIPTION ^

```
findShortestVector Returns the length of the shortest vector in samples
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Returns the length of the shortest vector in samples

CALL:      lmin = findShortestVector(as)

VERSION:    $Id: findShortestVector.m,v 1.5 2008/08/01 13:19:42 ingo Exp $

HISTORY:    14-05-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 **len** LEN overloads the length operator for Analysis objects. Length of the data samples.

This function is called by:

-  **ao** AO analysis object class constructor.
-  **xcorr** XCORR makes cross-correlation estimates of the time-series

SOURCE CODE ^

```
0001 % findShortestVector Returns the length of the shortest vector in samples
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Returns the length of the shortest vector in samples
0005 %
0006 % CALL:      lmin = findShortestVector(as)
0007 %
0008 % VERSION:    $Id: findShortestVector.m,v 1.5 2008/08/01 13:19:42 ingo Exp $
0009 %
0010 % HISTORY:    14-05-07 M Hewitson
0011 %              Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014 function lmin = findShortestVector(as)
0015
0016     lmin = 1e20;
0017     for j=1:length(as)
0018         len_as = len(as(j));
0019         if len_as < lmin
0020             lmin = len_as;
0021         end
0022     end
0023
0024 end
0025
```


fq2fac

PURPOSE ^

FQ2FAC is a private function and is called by ngconv.m which can be found in the

SYNOPSIS ^

function polzero = fq2fac(f,q)

DESCRIPTION ^

FQ2FAC is a private function and is called by ngconv.m which can be found in the folder 'noisegenerator'. It calculates polynomial coefficients from given poles and Zeros.

Inputs (from ngconv.m):

- f : frequency of apole or zero
- q : quality factor of a pole or zero

Outputs:

- polzero: a vector of resulting polynomial coefficients

A Monsky 24-07-07

\$Id: fq2fac.m,v 1.2 2008/08/01 13:19:42 ingo Exp \$

CROSS-REFERENCE INFORMATION ^

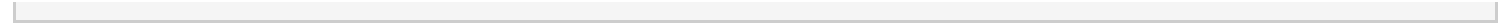
This function calls:

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FQ2FAC is a private function and is called by ngconv.m which can be found in the
0002 % folder 'noisegenerator'.
0003 % It calculates polynomial coefficients from given poles and Zeros.
0004 %
0005 % Inputs (from ngconv.m):
0006 %     - f : frequency of apole or zero
0007 %     - q : quality factor of a pole or zero
0008 %
0009 % Outputs:
0010 %     - polzero: a vector of resulting polynomial coefficients
0011 %
0012 % A Monsky 24-07-07
0013 %
0014 % $Id: fq2fac.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0015 %
0016
0017 function polzero = fq2fac(f,q)
0018
0019     n = length(f);
0020     polzero = zeros(n,3);
0021     for i = 1:n
0022         if isnan(q(i))
0023             polzero(i,1:2) = [1 1/(2*pi*f(i))];
0024         else
0025             polzero(i,1:3) = [1 1/(2*pi*f(i)*q(i)) 1/((2*pi*f(i))*(2*pi*f(i)))];
0026         end
0027     end
0028
0029 end
0030
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromDataInMAT

PURPOSE ^

FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object

SYNOPSIS ^

function obj = fromDataInMAT(obj, data, pli)

DESCRIPTION ^

```
FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromDataInMAT
DESCRIPTION:   Convert a saved data-array into an AO with a tsdata-object
CALL:         obj = fromLISO(obj, data-array, plist)
PARAMETER:    data-array: data-array
               plist:      plist-object (must contain the filename)
VERSION:      $Id: fromDataInMAT.m,v 1.1 2008/08/22 14:17:26 ingo Exp $
HISTORY:      22-03-2008 M Hewitson
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **find** FIND particular samples that satisfy the input query and return a new AO.
-  **fs** FS Get the data property 'fs'.
-  **x** X Get the data property 'x'.

This function is called by:

-  **ao** AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromDataInMAT
0005 %
0006 % DESCRIPTION:   Convert a saved data-array into an AO with a tsdata-object
0007 %
0008 % CALL:         obj = fromLISO(obj, data-array, plist)
0009 %
0010 % PARAMETER:    data-array: data-array
0011 %               plist:      plist-object (must contain the filename)
0012 %
0013 % VERSION:      $Id: fromDataInMAT.m,v 1.1 2008/08/22 14:17:26 ingo Exp $
0014 %
0015 % HISTORY:      22-03-2008 M Hewitson
0016 %               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 %
0020 function obj = fromDataInMAT(obj, data, pli)
```

```

0021
0022     VERSION = '$Id: fromDataInMAT.m,v 1.1 2008/08/22 14:17:26 ingo Exp $';
0023
0024     ii = obj.getInfo(class(obj), 'From MAT File');
0025
0026     % Set the method version string in the minfo object
0027     ii.setMversion([VERSION '-->' ii.mversion]);
0028
0029     % Then we try for a numerical data set in
0030     % the first numerical field we come to
0031     fnames = fieldnames(data);
0032     for jj=1:length(fnames)
0033         if isnumeric(data.(fnames{jj}))
0034             % get the data from here
0035             data = data.(fnames{jj});
0036             ss = size(data);
0037             if ss(1) > 2
0038                 warning('!!! I can only handle data files with two columns: time and amplitude.
Additional columns are ignored.');
```

```

0039             end
0040             if ss(1) > ss(2)
0041                 data = data.';
0042             end
0043             if ss(1) < 2
0044                 x = data(1,:);
0045                 fs = 1;
0046                 data_obj = tsdata(x, fs);
0047             else
0048                 % t from column 1
0049                 t = data(1,:);
0050                 % x from column 2
0051                 x = data(2,:);
0052                 data_obj = tsdata(t,x);
0053             end
0054
0055             obj.data = data_obj;
0056
0057             % ignore the other fields
0058             warning('!!! Found a numeric field called %. Ignoring all others.', fnames{jj});
0059
0060             obj.setName(find(pli, 'filename'), 'internal');
0061             obj.addHistory(ii, pli, [], []);
0062             break;
0063         end
0064     end
0065
0066 end
0067

```

fromDatafile

PURPOSE ^

FROMDATAFILE Construct an ao from filename AND parameter list

SYNOPSIS ^

function a = fromDatafile(ain, pli)

DESCRIPTION ^

```
FROMDATAFILE Construct an ao from filename AND parameter list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromDatafile

DESCRIPTION:   Construct an ao from filename AND parameter list

CALL:         a = fromFilenameAndPlist(a, pli)

PARAMETER:    a:   empty ao-object
               pli: plist-object (must contain the filename)

VERSION:      $Id: fromDatafile.m,v 1.5 2008/09/07 18:08:03 hewitson Exp $

HISTORY:      07-05-2007 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [max](#) MAX computes the maximum value of the data in the AO.
-  [min](#) MIN computes the minimum value of the data in the AO.
-  [sort](#) SORT the values in the AO.
-  [sum](#) SUM computes the sum of the data in the AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  function lines = numlines(fid)
-  function f_data = robustRead(fid, f_data, columns, orig_cols)

SOURCE CODE ^

```
0001 % FROMDATAFILE Construct an ao from filename AND parameter list
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromDatafile
0005 %
0006 % DESCRIPTION:   Construct an ao from filename AND parameter list
```

```

0007 %
0008 % CALL:      a = fromFilenameAndPlist(a, pli)
0009 %
0010 % PARAMETER:  a:   empty ao-object
0011 %             pli: plist-object (must contain the filename)
0012 %
0013 % VERSION:    $Id: fromDatafile.m,v 1.5 2008/09/07 18:08:03 hewitson Exp $
0014 %
0015 % HISTORY:    07-05-2007 Hewitson
0016 %             Creation
0017 %
0018 %%%%%%%%%%%
0019
0020 function a = fromDatafile(ain, pli)
0021
0022     utils.helper.msg(utils.const.msg.PROC1, 'constructing from filename and/or plist');
0023
0024     VERSION = '$Id: fromDatafile.m,v 1.5 2008/09/07 18:08:03 hewitson Exp $';
0025
0026     % get AO info
0027     mi = ao.getInfo('ao', 'From ASCII File');
0028
0029     % Set the method version string in the minfo object
0030     mi.setMversion([VERSION '-->' mi.mversion]);
0031
0032     % Get filename
0033     file_name = find(pli, 'filename');
0034
0035     [pathstr, f_name, ext] = fileparts(file_name);
0036
0037     %%%%%%%%%% Get default parameter list %%%%%%%%%%
0038     dpl = ao.getDefaultPlist('From ASCII File');
0039     pl = combine(pli, dpl);
0040
0041     pl = pset(pl, 'filename', [f_name ext]);
0042     pl = pset(pl, 'filepath', pathstr);
0043
0044     data_type = find (pl, 'type');
0045     columns = find (pl, 'columns');
0046     maxLines = find (pl, 'maxlines');
0047     comment_char = find (pl, 'comment_char');
0048     use_fs = find (pl, 'use_fs');
0049     a = [];
0050     orig_col = columns; % necessary fro the histroy
0051
0052     %%%%%%%%%% read file %%%%%%%%%%
0053     [fid,msg] = fopen (file_name, 'r');
0054     if (fid < 0)
0055         error ('### can not open file: %s \n\n### error msg:',file_name, msg);
0056     end
0057
0058     %%%%%%%%%% create scan format: '%f %f %f %f %f %*[\n]' %%%%%%%%%%
0059     maxcol = max(columns);
0060     scan_format = '';
0061     read_col = 0;
0062     sort_col = sort(columns);
0063
0064     %%%%%%%%%% Get/Count max number of lines %%%%%%%%%%
0065     if isempty(maxLines) || maxLines < 0
0066         maxLines = 1e20;
0067     end
0068     if maxLines == 1e20
0069         maxLines = numlines(fid);
0070         utils.helper.msg(utils.const.msg.PROC2, 'Counting lines: %d', maxLines);
0071     end
0072     fseek(fid, 0, 'bof');
0073
0074     %%%%%%%%%% Read data %%%%%%%%%%
0075     readlines = min(50000, maxLines);
0076     nlines = 0;
0077
0078     %% Based on skipping the not used columns we have to transform the columns.
0079     %% We must transform the columns [ 2 5 2 6 5 7] to [ 1 2 1 3 2 4]
0080     %% In each loop we have to replace the corresponding value. In the first loop
0081     %% the first minimum, in the second loop the second minimum, ... with the
0082     %% current loop number.
0083     for j=1:max(columns)
0084         if ismember(j, columns)
0085             scan_format = [scan_format '%n '];
0086             read_col = read_col + 1;
0087             replace = min(sort_col);
0088
0089             columns(columns==replace) = read_col;
0090             sort_col(sort_col==replace) = [];
0091         else
0092             scan_format = [scan_format '%*n '];

```



```

0093     end
0094 end
0095 scan_format = [deblank(scan_format) '%*[^\\n]'];
0096
0097 %%% preallocate data array
0098 f_data = zeros(maxLines, read_col);
0099
0100 if strcmpi(find(pl, 'Robust'), 'yes')
0101     f_data = robustRead(fid, f_data, columns, orig_col);
0102
0103 else
0104
0105     %%% Look for the first line of data
0106     while ~feof(fid)
0107         f = deblank(fgetl(fid));
0108         if ~isempty(f)
0109             if f(1) ~= comment_char
0110                 break;
0111             end
0112         end
0113     end
0114
0115     %%% Scan it to find how many columns we have in the file
0116     C = textscan(f, scan_format, 1, ...
0117         'CommentStyle', comment_char, ...
0118         'CollectOutput', 1);
0119     if any(isnan(C{:}))
0120         error('### Error in file format. Perhaps you specified more columns than the file
contains?');
0121     end
0122
0123     fseek(fid, 0, 'bof');
0124     %%% read file to end
0125     while ~feof(fid) && nlines < maxLines
0126         C = textscan(fid, scan_format, readlines, ...
0127             'CommentStyle', comment_char, ...
0128             'CollectOutput', 1);
0129         f_data(nlines+1:nlines+size(C{1},1), :) = C{1};
0130         nlines = nlines + length(C{1});
0131
0132         if isempty(C{1})
0133             error('\n### There are no data.\n### Did you use the right comment character?
\n### The current comment character is: [%s]\n### Use a parameter list with the
parameter:\n### plist('comment_char', '%*')', comment_char);
0134         end
0135     end
0136
0137     utils.helper.msg(utils.const.msg.PROC2, 'read %09d lines of %09d', nlines,
maxLines);
0138 end
0139 fclose(fid);
0140
0141 %%% get only the data we want
0142 if size(f_data,1) > nlines
0143     f_data = f_data(1:nlines, :);
0144 end
0145
0146 %%% %%% %%% %%% %%% Create for each column pair the data object %%% %%% %%% %%% %%%
0147 if isempty(use_fs)
0148
0149     %%% %%% %%% %%% %%% The numbers in columns must be straight %%% %%% %%% %%% %%%
0150     if mod(length(columns),2) ~= 0
0151         error('### the numbers in columns must be straight');
0152     end
0153
0154     for lauf = 1:length(columns)/2
0155
0156         data_x_axes = f_data(:, columns(lauf*2-1));
0157         data_y_axes = f_data(:, columns(lauf*2));
0158
0159         % create data object corresponding to the parameter list
0160         ao_data = [];
0161         switch data_type
0162             case 'tsdata'
0163                 ao_data = tsdata( data_x_axes, data_y_axes);
0164             case 'fsdata'
0165                 ao_data = fsdata( data_x_axes, data_y_axes);
0166             case 'cdata'
0167                 error ('### please code me up')
0168             case 'xydata'
0169                 ao_data = xydata( data_x_axes, data_y_axes);
0170             otherwise
0171                 error('### unknown data type '%s'', data_type);
0172         end
0173     end
0174

```

```

0175 aa = ao(ao_data);
0176 aa.setName(sprintf('%s_%02d_%02d', file_name, orig_col(lauf*2-1),
orig_col(lauf*2)), 'internal');
0177 % Add history
0178 pl = pl.pset('columns', [orig_col(lauf*2-1) orig_col(lauf*2)]);
0179 aa.addHistory(mi, pl, [], []);
0180
0181 % Set the description from the parameter list or from the file
0182 description_pl = find(pl, 'description');
0183 if ~isempty(description_pl)
0184 aa.description = description_pl;
0185 end
0186
0187 a = [a aa];
0188
0189 end
0190
0191 %%%%%%%%%%% Create for each column AND fs a data object %%%%%%%%%%%
0192 else % isempty(use_fs)
0193
0194 for lauf = 1:length(columns)
0195
0196 data_y_axes = f_data(:, columns(lauf));
0197
0198 % create data object corresponding to the parameter list
0199 ao_data = [];
0200 switch data_type
0201 case 'tsdata'
0202 ao_data = tsdata(data_y_axes, use_fs);
0203 case 'fsdata'
0204 ao_data = fsdata(data_y_axes, use_fs);
0205 case 'cdata'
0206 error('### please code me up')
0207 case 'xydata'
0208 error('### please code me up')
0209 otherwise
0210 error('### unknown data type '%s'', data_type);
0211 end
0212 aa = ao(ao_data);
0213 aa.setName(sprintf('%s_%02d', file_name, orig_col(lauf)), 'internal');
0214 pl = pl.pset('columns', orig_col(lauf));
0215 aa.addHistory(mi, pl, [], []);
0216
0217 % Set the description from the parameter list or from the file
0218 description_pl = find(pl, 'description');
0219 if ~isempty(description_pl)
0220 aa.description = description_pl;
0221 end
0222
0223 a = [a aa];
0224
0225 end
0226
0227 end
0228
0229 %%%%%%%%%%% set fields of the AO from the parameter list %%%%%%%%%%%
0230 for ii = 1:length(a)
0231
0232 for jj = 1:pli.nparams
0233
0234 cmd = ['set' pli.params(jj).key(1) lower(pli.params(jj).key(2:end))];
0235
0236 if ismethod(a(ii), cmd)
0237 if iscell(pli.params(jj).val) && length(pli.params(jj).val) == length(a)
0238 %%% Set the value in the cell
0239 feval(cmd, a(ii), pli.params(jj).val{ii}, 'internal');
0240 %%% Reset the value in the plistUset to this single value
0241 a(ii).hist.plistUsed.pset(pli.params(jj).key, pli.params(jj).val{ii});
0242 else
0243 %%% Set the value
0244 feval(cmd, a(ii), pli.params(jj).val, 'internal');
0245 end
0246 end
0247
0248 end
0249
0250 end % for-loop over all ao's
0251
0252 end
0253
0254 %%%%%%%%%%% %%%%%%%%%%%
0255 % Local Functions %
0256 %%%%%%%%%%% %%%%%%%%%%%
0257
0258 %%%%%%%%%%% %%%%%%%%%%%
0259 %

```

```

0260 % FUNCTION:      numlines
0261 %
0262 % SYNTAX:         count = numlines(fid);
0263 %
0264 % DESCRIPTION:    Number of lines in an ASCII file
0265 %
0266 % HISTORY:        02-08-2002 Peter Acklam, CSSM post
0267 %                  Creation.
0268 %
0269 %%%%%%%%%%%
0270
0271 function lines = numlines(fid)
0272
0273     lines = 0;                                % number of lines in file
0274     nlchr = uint8(sprintf('\n'));              % newline chr as uint8
0275     bsize = 4 * 256 * 8192;                    % block size to read
0276
0277     while ~feof(fid)
0278         block = fread(fid, bsize, '*uint8');
0279         lines = lines + sum(block == nlchr);
0280     end
0281     if ~isempty(block)                        % in case file is empty
0282         lines = lines + double(block(end) ~= nlchr);
0283     end
0284 end
0285
0286 % A robust and slow data reader
0287 function f_data = robustRead(fid, f_data, columns, orig_cols)
0288
0289     cols = unique(columns);
0290     ocols = unique(orig_cols);
0291     Nline = 1;
0292     while ~feof(fid)
0293         % read and parse line
0294         tokens = sscanf(fgets(fid), '%f');
0295         % parse tokens
0296         if ~isempty(tokens)
0297             f_data(Nline, cols) = tokens(ocols);
0298             if mod(Nline, 1000) == 0
0299                 utils.helper.msg(utils.const.msg.PROC2, 'lines read: %d', Nline);
0300             end
0301             Nline = Nline + 1;
0302         end
0303     end
0304
0305     % drop empty lines
0306     f_data = f_data(1:Nline-1, :);
0307
0308 end

```

fromFSfcn

PURPOSE ^

FROMFSFCN Construct an ao from a fs-function string

SYNOPSIS ^

function a = fromFSfcn(a, pli)

DESCRIPTION ^

```
FROMFSFCN Construct an ao from a fs-function string
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromFSfcn

DESCRIPTION: Construct an ao from a fs-function string

CALL:         a = fromFSfcn(a, pl)

PARAMETER:    pl:      Parameter list object

HISTORY:      07-05-2007 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [log10](#) LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMFSFCN Construct an ao from a fs-function string
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromFSfcn
0005 %
0006 % DESCRIPTION: Construct an ao from a fs-function string
0007 %
0008 % CALL:         a = fromFSfcn(a, pl)
0009 %
0010 % PARAMETER:    pl:      Parameter list object
0011 %
0012 %
0013 % HISTORY:      07-05-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function a = fromFSfcn(a, pli)
0018
0019     import utils.const.*
0020
```

```

0021 VERSION = '$Id: fromFSfcn.m,v 1.12 2008/09/07 18:08:37 hewitson Exp $';
0022 % get AO info
0023 ii = ao.getInfo('ao', 'From Frequency-series Function');
0024
0025 % Set the method version string in the minfo object
0026 ii.setMversion([VERSION '-->' ii.mversion]);
0027
0028 % Add default values
0029 pl = combine(pli, ii.plists);
0030 f = find(pl, 'f');
0031 randn_state = find(pl, 'rand_state');
0032
0033 % If the randn state is in the parameter list then set the randn function to
0034 % this state
0035 if ~isempty(randn_state)
0036     randn('state', randn_state);
0037 else % Store the state of the random function in the parameter list
0038     randn_state = randn('state');
0039     pl.append('rand_state', randn_state);
0040 end
0041
0042 if isempty(f)
0043     utils.helper.msg(msg.PROC2, 'generating f vector');
0044     f1 = find(pl, 'f1');
0045     f2 = find(pl, 'f2');
0046     nf = find(pl, 'nf');
0047     scale = find(pl, 'scale');
0048     switch scale
0049     case 'log'
0050         f = logspace(log10(f1), log10(f2), nf);
0051     case 'lin'
0052         f = linspace(f1, f2, nf);
0053     otherwise
0054         error('### Unknown frequency scale specified');
0055     end
0056 end
0057
0058 % Get the function
0059 fcn = find(pl, 'fsfcn');
0060
0061 % make y data
0062 y = eval([fcn ';' ]);
0063
0064 fs = fsdata(f, y);
0065 fs.setXunits('Hz');
0066
0067 % Make an analysis object
0068 a.data = fs;
0069 % Add history
0070 a.addHistory(ii, pl, [], []);
0071
0072 end
0073

```

fromFcn

PURPOSE ^

FROMFCN Construct an ao from a function string

SYNOPSIS ^

function a = fromFcn(a, pli)

DESCRIPTION ^

```
FROMFCN Construct an ao from a function string
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromFcn
DESCRIPTION:   Construct an ao from a function string
CALL:         a = fromFcn(a, pl)
PARAMETER:    pl: Parameter list object
HISTORY:      07-05-2007 Hewitson
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMFCN Construct an ao from a function string
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromFcn
0005 %
0006 % DESCRIPTION:   Construct an ao from a function string
0007 %
0008 % CALL:         a = fromFcn(a, pl)
0009 %
0010 % PARAMETER:    pl: Parameter list object
0011 %
0012 %
0013 % HISTORY:      07-05-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function a = fromFcn(a, pli)
0018
0019     VERSION = '$Id: fromFcn.m,v 1.11 2008/09/04 13:37:14 ingo Exp $';
0020     % get AO info
0021     ii = ao.getInfo('ao', 'From Function');
0022
0023     % Set the method version string in the minfo object
0024     ii.setMversion([VERSION '-->' ii.mversion]);
0025
```

```

0026 % Add default values
0027 pl = combine(pli, ii.plists);
0028
0029 fcn      = find(pl, 'fcn');
0030 randn_state = find(pl, 'rand_state');
0031
0032 % If the randn state is in the parameter list then set the randn function to
0033 % this state
0034 if ~isempty(randn_state)
0035     randn('state', randn_state);
0036 else % Store the state of the random function in the parameter list
0037     randn_state = randn('state');
0038     pl.append('rand_state', randn_state);
0039 end
0040
0041 % find all other parameters and try to evaluate them
0042 for jj=1:nparams(pl)
0043     p = pl.params(jj);
0044     if ~strcmpi(fcn, p.val)
0045         % try upper case replacement
0046         fcn = strrep(fcn, p.key, mat2str(p.val));
0047         % try lower case replacement
0048         fcn = strrep(fcn, lower(p.key), mat2str(p.val));
0049     end
0050 end
0051
0052 % Set data of analysis object
0053 a.data = cdata(eval(fcn));
0054 % Add history
0055 a.addHistory(ii, pl, [], []);
0056
0057 end
0058

```

fromPolyval

PURPOSE ^

FROMPOLYVAL Construct a time-series ao from polynomial coefficients

SYNOPSIS ^

function a = fromPolyval(a, pli)

DESCRIPTION ^

```
FROMPOLYVAL Construct a time-series ao from polynomial coefficients
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromPolyval

DESCRIPTION:   Construct a time-series ao from polynomial coefficients

CALL:         a = fromPolyval(a, vals)

PARAMETER:    pl:      plist containing 'polyval', 'Nsecs', 'fs', or 't'

HISTORY:      07-05-2007 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMPOLYVAL Construct a time-series ao from polynomial coefficients
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromPolyval
0005 %
0006 % DESCRIPTION:   Construct a time-series ao from polynomial coefficients
0007 %
0008 % CALL:         a = fromPolyval(a, vals)
0009 %
0010 % PARAMETER:    pl:      plist containing 'polyval', 'Nsecs', 'fs', or 't'
0011 %
0012 %
0013 % HISTORY:      07-05-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function a = fromPolyval(a, pli)
0019
0020     VERSION = '$Id: fromPolyval.m,v 1.9 2008/09/04 13:37:14 ingo Exp $';
0021     % get AO info
0022     ii = ao.getInfo('ao', 'From Polynomial');
```



```
0023
0024 % Set the method version string in the minfo object
0025 ii.setMversion([VERSION '-->' ii.mversion]);
0026
0027 % Add default values
0028 pl = combine(pli, ii.plists);
0029
0030 coeffs = find(pl, 'polyval');
0031 Nsecs = find(pl, 'Nsecs');
0032 fs = find(pl, 'fs');
0033 t = find(pl, 't');
0034
0035 % Check t vector
0036 if isempty(t)
0037     if isempty(Nsecs) || isempty(fs)
0038         error('### Please provide either ''Nsecs'' and ''fs'', or ''t'' for polyval
constructor.');
```

```
0039     end
0040     t = linspace(0, Nsecs - 1/fs, Nsecs*fs);
0041 end
0042
0043 y = polyval(coeffs,t);
0044
0045 % Make a tsdata object
0046 ts = tsdata(t, y);
0047 ts.setXunits('s');
0048 ts.setYunits('V');
0049
0050 % Make an analysis object
0051 a.data = ts;
0052 a.addHistory(ii, pl, [], []);
0053
0054 end
0055
```

fromPzmodel

PURPOSE ^

FROMPZMODEL Construct a time-series ao from polynomial coefficients

SYNOPSIS ^

function a = fromPzmodel(a, pli)

DESCRIPTION ^

```
FROMPZMODEL Construct a time-series ao from polynomial coefficients
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromPzmodel

DESCRIPTION:   Construct a time-series ao from polynomial coefficients

CALL:         a = fromPzmodel(a, pl)

PARAMETER:    pl:      plist containing 'pzmodel', 'Nsecs', 'fs'

HISTORY:      14-05-2008 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **find** FIND particular samples that satisfy the input query and return a new AO.
-  **fs** FS Get the data property 'fs'.
-  **x** X Get the data property 'x'.
-  **y** Y Get the data property 'y'.

This function is called by:

-  **ao** AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMPZMODEL Construct a time-series ao from polynomial coefficients
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromPzmodel
0005 %
0006 % DESCRIPTION:   Construct a time-series ao from polynomial coefficients
0007 %
0008 % CALL:         a = fromPzmodel(a, pl)
0009 %
0010 % PARAMETER:    pl:      plist containing 'pzmodel', 'Nsecs', 'fs'
0011 %
0012 %
0013 % HISTORY:      14-05-2008 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function a = fromPzmodel(a, pli)
0018
0019     VERSION = '$Id: fromPzmodel.m,v 1.11 2008/09/07 11:09:50 hewitson Exp $';
0020
```

```

0021 % get AO info
0022 ii = ao.getInfo('ao', 'From Pzmodel');
0023
0024 % Set the method version string in the minfo object
0025 ii.setMversion([VERSION '-->' ii.mversion]);
0026
0027 % Add default values
0028 pl = combine(pli, ii.plist);
0029
0030 pzm = find(pl, 'pzmodel');
0031 nsecs = find(pl, 'Nsecs');
0032 fs = find(pl, 'fs');
0033 ndigits = find(pl, 'ndigits');
0034
0035 % Build t vector
0036 if isempty(nsecs) || nsecs == 0
0037     error('### Please provide ''Nsecs'' for pzmodel constructor.');
```

```

0038 end
0039 if isempty(fs) || fs == 0
0040     error('### Please provide ''fs'' for pzmodel constructor.');
```

```

0041 end
0042 % t = linspace(0, nsecs - 1/fs, nsecs*fs);
0043
0044 % Run noise generator
0045 % conversion
0046 disp(' - Filter coefficients are calculated from input pzmodel.');
```

```

0047 [num, den] = ao.ngconv(pzm);
0048
0049 % create matrices
0050 toolboxinfo = ver('symbolic');
```

```

0051
0052 if isempty(toolboxinfo)
0053     disp('the time series is calculated without the symbolic math toolbox')
0054     disp(' - Matrices are calculated from evaluated denominator coefficients.');
```

```

0055 [Tinit, Tprop, E] = ao.ngsetup(den, fs);
0056 else
0057     disp('the time series is calculated using the symbolic math toolbox')
0058     disp(' - Matrices are calculated from evaluated denominator coefficients.');
```

```

0059 if isempty(ndigits)
0060     ndigits = 32;
0061     warning('### set number of digits to 32!')
```

```

0062 end
0063 [Tinit, Tprop, E] = ao.ngsetup_vpa(den, fs, ndigits);
0064 end
0065
0066 % set state vector
0067 disp(' - Since there is no given state vector it will be calculated.');
```

```

0068
0069 % make initial state vector
0070 y = ao.nginit(Tinit);
0071
0072 % propagate to make noise vector
0073 [x, yo] = ao.ngprop(Tprop, E, num, y, fs*nsecs);
0074
0075 % build variables into data object
0076 t = x*pzm.gain;
0077 a.data = tsdata(t, fs);
0078 a.setName(sprintf('noisegen(%s)', pzm.name), 'internal');
```

```

0079
0080 % Set units to a sensible default
0081 a.setXunits('s', 'internal');
```

```

0082 a.setYunits('V', 'internal');
```

```

0083
0084 % Add history
0085 a.addHistory(ii, pl, [], pzm.hist);
0086
0087 end
0088
0089

```

fromSpecWin

PURPOSE ^

FROMSPECWIN Construct an ao from a Spectral window

SYNOPSIS ^

function a = fromSpecWin(a, pli)

DESCRIPTION ^

```
FROMSPECWIN Construct an ao from a Spectral window
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromSpecWin

DESCRIPTION:   Construct an ao from a Spectral window

CALL:         a = fromSpecWin(a, win)

PARAMETER:    win:      Spectral window object

HISTORY:      07-05-2007 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMSPECWIN Construct an ao from a Spectral window
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromSpecWin
0005 %
0006 % DESCRIPTION:   Construct an ao from a Spectral window
0007 %
0008 % CALL:         a = fromSpecWin(a, win)
0009 %
0010 % PARAMETER:    win:      Spectral window object
0011 %
0012 %
0013 % HISTORY:      07-05-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function a = fromSpecWin(a, pli)
0018
0019     VERSION = '$Id: fromSpecWin.m,v 1.10 2008/09/04 13:37:14 ingo Exp $';
0020
0021     % get AO info
0022     ii = ao.getInfo('ao', 'From Window');
0023
0024     % Set the method version string in the minfo object
0025     ii.setMversion([VERSION '-->' ii.mversion]);
```

```
0026
0027     if isa(pli, 'plist')
0028         % Add default values
0029         pl = combine(pli, ii.plists);
0030
0031         win = find(pl, 'win');
0032     elseif isa(pli, 'specwin')
0033         win = pli;
0034         pl = plist('win', win);
0035     else
0036         error('### Unknow usage. ');
0037     end
0038
0039     % Make a cdata object
0040     a.data = cdata(win.win);
0041     a.setName(win.type, 'internal');
0042
0043     % Add history
0044     a.addHistory(ii, pl, [], []);
0045
0046 end
0047
0048
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromTSfcn

PURPOSE ^

FROMTSFCN Construct an ao from a ts-function string

SYNOPSIS ^

function a = fromTSfcn(a, pli)

DESCRIPTION ^

FROMTSFCN Construct an ao from a ts-function string

FUNCTION: fromTSfcn

DESCRIPTION: Construct an ao from a ts-function string

CALL: a = fromTSfcn(pl)

PARAMETER: pl: Parameter list object

HISTORY: 07-05-2007 Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

-  **find** FIND particular samples that satisfy the input query and return a new AO.
-  **fs** FS Get the data property 'fs'.
-  **sum** SUM computes the sum of the data in the AO.
-  **t0** T0 Get the data property 't0'.
-  **y** Y Get the data property 'y'.

This function is called by:

-  **ao** AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMTSFCN Construct an ao from a ts-function string
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION: fromTSfcn
0005 %
0006 % DESCRIPTION: Construct an ao from a ts-function string
0007 %
0008 % CALL: a = fromTSfcn(pl)
0009 %
0010 % PARAMETER: pl: Parameter list object
0011 %
0012 %
0013 % HISTORY: 07-05-2007 Hewitson
0014 % Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function a = fromTSfcn(a, pli)
0018
```

```

0019 VERSION = '$Id: fromTSfcn.m,v 1.12 2008/09/04 13:37:14 ingo Exp $';
0020
0021 % get AO info
0022 ii = ao.getInfo('ao', 'From Time-series Function');
0023
0024 % Set the method version string in the minfo object
0025 ii.setMversion([VERSION '-->' ii.mversion]);
0026
0027 % Add default values
0028 pl = combine(pli, ii.plist);
0029
0030 nsecs      = find(pl, 'nsecs');
0031 fs         = find(pl, 'fs');
0032 fcn        = find(pl, 'tsfcn');
0033 randn_state = find(pl, 'rand_state');
0034 t0         = find(pl, 't0');
0035
0036 if isempty(t0)
0037     t0 = time(0);
0038 end
0039
0040 % construct tsdata
0041 t = linspace(0, nsecs-1/fs, nsecs*fs);
0042
0043 % make y data
0044
0045 % If the randn state is in the parameter list then set the randn function to
0046 % this state
0047 if ~isempty(randn_state)
0048     randn('state', randn_state);
0049 else % Store the state of the random function in the parameter list
0050     randn('state', sum(100*clock));
0051     randn_state = randn('state');
0052     % we must make a copy here, other wise all other calls
0053     % using the input plist will get the same random state
0054     pl.append('rand_state', randn_state);
0055 end
0056
0057 y = eval([fcn ';']);
0058
0059 ts = tsdata(t, y);
0060 ts.setXunits('s');
0061 ts.setYunits('V');
0062 ts.setT0(t0);
0063
0064 % Make an analysis object
0065 a.data = ts;
0066 % Add history
0067 a.addHistory(ii, pl, [], []);
0068
0069 end
0070
0071

```

fromVals

PURPOSE ^

FROMVALS Construct an ao from a value set

SYNOPSIS ^

function a = fromVals(a, pli)

DESCRIPTION ^

```
FROMVALS Construct an ao from a value set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromVals
DESCRIPTION:   Construct an ao from a value set
CALL:         a = aoFromVals(a, vals)
PARAMETER:    vals:      Constant values
HISTORY:      07-05-2007 Hewitson
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [find](#) FIND particular samples that satisfy the input query and return a new AO.

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMVALS Construct an ao from a value set
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromVals
0005 %
0006 % DESCRIPTION:   Construct an ao from a value set
0007 %
0008 % CALL:         a = aoFromVals(a, vals)
0009 %
0010 % PARAMETER:    vals:      Constant values
0011 %
0012 %
0013 % HISTORY:      07-05-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function a = fromVals(a, pli)
0018
0019     VERSION = '$Id: fromVals.m,v 1.14 2008/09/04 13:37:14 ingo Exp $';
0020
0021     % get AO info
0022     ii = ao.getInfo('ao', 'From Vals');
0023
0024     % Set the method version string in the minfo object
0025     ii.setMversion([VERSION '-->' ii.mversion]);
```



```

0026
0027 % Combine input plist with default values
0028 pl = combine(pli, ii.plist);
0029
0030 % Get values from the plist
0031 vals = find(pl, 'vals');
0032 xvals = find(pl, 'xvals');
0033 yvals = find(pl, 'yvals');
0034 N = find(pl, 'N');
0035
0036 % If the user specified no repeats, we go for 1 repeat.
0037 if isempty(N)
0038     N = 1;
0039 end
0040
0041 %----- Now decide what to construct
0042
0043 % We have x and y values
0044 if ~isempty(xvals) && ~isempty(yvals)
0045     if ischar(xvals)
0046         try
0047             xvals = eval(xvals);
0048         catch
0049             error('Could not convert char array to numeric array.');

```

fromWaveform

PURPOSE ^

FROMWAVEFORM Construct an ao from a waveform

SYNOPSIS ^

function a = fromWaveform(a, pli)

DESCRIPTION ^

```
FROMWAVEFORM Construct an ao from a waveform
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromWaveform

DESCRIPTION:   Construct an ao from a waveform

CALL:         a = fromWaveform(pl)

PARAMETER:    pl:      Parameter list object

HISTORY:      07-05-2007 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [t0](#) T0 Get the data property 't0'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % FROMWAVEFORM Construct an ao from a waveform
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromWaveform
0005 %
0006 % DESCRIPTION:   Construct an ao from a waveform
0007 %
0008 % CALL:         a = fromWaveform(pl)
0009 %
0010 % PARAMETER:    pl:      Parameter list object
0011 %
0012 %
0013 % HISTORY:      07-05-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function a = fromWaveform(a, pli)
0018
0019     VERSION = '$Id: fromWaveform.m,v 1.10 2008/09/04 13:37:14 ingo Exp $';
0020
```

```

0021 % get AO info
0022 ii = ao.getInfo('ao', 'From Waveform');
0023
0024 % Set the method version string in the minfo object
0025 ii.setMversion([VERSION '-->' ii.mversion]);
0026
0027 % Add default values
0028 pl = combine(pli, ii.plists);
0029
0030 nsecs = find(pl, 'nsecs');
0031 fs = find(pl, 'fs');
0032 t0 = find(pl, 't0');
0033 if isempty(t0)
0034     t0 = time(0);
0035 end
0036
0037 waveform = lower(find(pl, 'waveform'));
0038 switch waveform
0039     %----- Sine Wave
0040     case {'sine wave', 'sinewave', 'sine-wave'}
0041         ampl = find(pl, 'A');
0042         freq = find(pl, 'f');
0043         phi = find(pl, 'phi');
0044
0045         if length(ampl) ~= length(freq) || length(freq) ~= length(phi) || length(ampl) ~=
length(phi)
0046             error('### Provide an Amplitude, frequency and phase for each sine-wave in the
series.');
```

```

0047         end
0048
0049         if isempty(ampl)
0050             ampl = 1;
0051         end
0052         if isempty(freq)
0053             freq = 1;
0054         end
0055         if isempty(phi)
0056             phi = 0;
0057         end
0058
0059         % build command
0060         tsfcn = '';
0061         for jj=1:length(ampl)
0062             amp = ampl(jj);
0063             fr = freq(jj);
0064             ph = phi(jj);
0065
0066             if jj==1
0067                 tsfcn = sprintf('%g*sin(2*pi*%g*t + %g*pi/180)', amp, fr, ph);
0068             else
0069                 tsfcn = [tsfcn sprintf(' + %g*sin(2*pi*%g*t + %g*pi/180)', amp, fr, ph)];
0070             end
0071         end
0072
0073         %----- Noise
0074         case 'noise'
0075             ntype = find(pl, 'type');
0076             if isempty(ntype)
0077                 ntype = 'Normal';
0078             end
0079
0080             randn_state = find(pl, 'rand_state');
0081
0082             switch lower(ntype)
0083                 case 'normal'
0084                     tsfcn = 'randn(size(t))';
0085                     % Set random state
0086                     if ~isempty(randn_state)
0087                         randn('state', randn_state);
0088                     else
0089                         randn_state = randn('state');
0090                         pl.append('rand_state', randn_state);
0091                     end
0092
0093                 case 'uniform'
0094                     tsfcn = 'rand(size(t))';
0095                     % Set random state
0096                     if ~isempty(randn_state)
0097                         rand('seed', randn_state);
0098                     else
0099                         randn_state = rand('seed');
0100                         pl.append('rand_state', randn_state);
0101                     end
0102
0103             end
0104         %----- Chirp

```

```

0105     case 'chirp'
0106         f0 = find(pl, 'f0');
0107         fe = find(pl, 'f1');
0108         te = find(pl, 't1');
0109         if isempty(f0)
0110             f0 = 0;
0111         end
0112         if isempty(fe)
0113             fe = fs/2;
0114         end
0115         if isempty(te)
0116             te = nsecs;
0117         end
0118         tsfcn = sprintf('chirp(t,%g,%g,%g)', f0, fe, te);
0119         %----- Gaussian pulse
0120     case {'gaussian pulse', 'gaussian-pulse'}
0121         fc = find(pl, 'f0');
0122         bw = find(pl, 'bw');
0123         if isempty(fc)
0124             fc = 1;
0125         end
0126         if isempty(bw)
0127             bw = fs/2;
0128         end
0129         tsfcn = sprintf('gauspuls(t,%g,%g)', fc, bw);
0130     case {'square wave', 'squarewave', 'square-wave'}
0131         freq = find(pl, 'f');
0132         duty = find(pl, 'duty');
0133         if isempty(freq)
0134             freq = 1;
0135         end
0136         if isempty(duty)
0137             duty = 50;
0138         end
0139         tsfcn = sprintf('square(2*pi*%g*t,%g)', freq, duty);
0140     case {'saw tooth', 'sawtooth', 'saw-tooth'}
0141         freq = find(pl, 'f');
0142         width = find(pl, 'width');
0143         if isempty(freq)
0144             freq = 1;
0145         end
0146         if isempty(width)
0147             width = 0.5;
0148         end
0149         tsfcn = sprintf('sawtooth(2*pi*%g*t,%g)', freq, width);
0150     end
0151
0152     % construct tsdata
0153     t = linspace(0, nsecs-1/fs, nsecs*fs);
0154     % make y data
0155     y = eval([tsfcn ';' ]);
0156
0157     ts = tsdata(t,y);
0158     ts.setXunits('S');
0159     ts.setYunits('V');
0160     ts.setT0(t0);
0161
0162     % Make an analysis object
0163     a.data = ts;
0164     a.setName(waveform, 'internal');
0165     % Add history
0166     a.addHistory(ii, pl, [], []);
0167
0168
0169 end
0170

```

fs

PURPOSE ^

FS Get the data property 'fs'.

SYNOPSIS ^

function varargout = fs(varargin)

DESCRIPTION ^

```
FS Get the data property 'fs'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the data property 'fs'.

CALL:      val = ao.fs();

INPUTS:    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('fs')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('fs', 'None')

VERSION:    $Id: fs.m,v 1.1 2008/09/05 13:55:17 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.
-  [dft](#) DFT computes the DFT of the input time-series at the requested frequencies.
-  [fft](#) FFT overloads the fft method for Analysis objects.
-  [filtfilt](#) FILTFILT overrides the filtfilt function for analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [findFsMax](#) findFsMax Returns the max Fs of a set of AOs
-  [fixfs](#) FIXFS resamples the input time-series to have a fixed sample rate.
-  [fngen](#) FNGEN creates an arbitrarily long time-series based on the input PSD.
-  [fromDataInMAT](#) FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object
-  [fromFSfcn](#) FROMFSFCN Construct an ao from a fs-function string
-  [fromPolyval](#) FROMPOLYVAL Construct a time-series ao from polynomial coefficients
-  [fromPzmodel](#) FROMPZMODEL Construct a time-series ao from polynomial coefficients
-  [fromTSfcn](#) FROMTSFCN Construct an ao from a ts-function string
-  [fromWaveform](#) FROMWAVEFORM Construct an ao from a waveform
-  [interp](#) INTERP interpolate the values in the input AO(s) at new values.
-  [join](#) JOIN multiple AOs into a single AO.
-  [linedetect](#) LINEDETECT find spectral lines in the ao/fsdata objects.

-  [ltf_plan](#) LTF_PLAN computes all input values needed for the LPSD and LTFE algorithms.
-  [mlpsd_m](#) MLPSD_M m-file only version of the LPSD algorithm
-  [mlpsd_mex](#) MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD
-  [mltfe](#) MLTFE compute log-frequency space TF
-  [ngsetup](#) NGSETUP is called by the function LTPDA_NOISEGEN
-  [ngsetup_vpa](#) ALGONAME = mfilename;
-  [psd](#) PSD makes power spectral density estimates of the time-series objects
-  [resample](#) RESAMPLE overloads resample function for AOs.
-  [select](#) SELECT select particular samples from the input AOs and return new AOs with only those samples.
-  [spikecleaning](#) spikecleaning detects and corrects possible spikes in analysis objects
-  [welch](#) WELCH Welch spectral estimation method.
-  [welchscale](#) WELCHSCALE scales the output of welch to be in the required units
-  [xcorr](#) XCORR makes cross-correlation estimates of the time-series
-  [xspect](#) XSPEC performs cross-spectral analysis of various forms.

SUBFUNCTIONS [^]

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE [^]

```

0001 % FS Get the data property 'fs'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the data property 'fs'.
0005 %
0006 % CALL:      val = ao.fs();
0007 %
0008 % INPUTS:    obj - must be a single data2D object.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %              >> ao.getInfo('fs')
0012 %
0013 %              Get information about a specified set-plot by calling:
0014 %              >> ao.getInfo('fs', 'None')
0015 %
0016 % VERSION:    $Id: fs.m,v 1.1 2008/09/05 13:55:17 ingo Exp $
0017 %
0018 % HISTORY:    27-05-2008 Diepholz
0019 %              Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = fs(varargin)
0024 %
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 obj = varargin{1};
0032 %
0033 if isprop(obj.data, 'fs')
0034     varargout{1} = obj.data.fs;
0035 else
0036     varargout{1} = [];
0037 end
0038 %
0039 end
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %              Local Functions
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:    getInfo

```

```
0048 %
0049 % DESCRIPTION: Get Info Object
0050 %
0051 % HISTORY:      11-07-07 M Hewitson
0052 %              Creation.
0053 %
0054 %%%%%%%%%%%
0055 %
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl    = [];
0060     else
0061         sets = {'Default'};
0062         pl    = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ao', '', utils.const.categories.internal, '$Id: fs.m,v 1.1
2008/09/05 13:55:17 ingo Exp $', sets, pl);
0066 end
0067 %%%%%%%%%%%
0068 %
0069 %
0070 % FUNCTION:      getDefaultPlist
0071 %
0072 % DESCRIPTION: Get Default Plist
0073 %
0074 % HISTORY:      11-07-07 M Hewitson
0075 %              Creation.
0076 %
0077 %%%%%%%%%%%
0078 %
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082
```

ltf_plan

PURPOSE ^

LTF_PLAN computes all input values needed for the LPSD and LTFE algorithms.

SYNOPSIS ^

function varargout = ltf_plan(varargin)

DESCRIPTION ^

```
LTF_PLAN computes all input values needed for the LPSD and LTFE algorithms.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTF_PLAN computes all input values needed for the LPSD and LTFE
             algorithms.

CALL:       [f, r, b, L, K] = ltf_plan(Ndata, fs, olap, bmin, Lmin, Jdes, Kdes)

INPUTS:     Ndata  - the length of the time-series to be processed
             fs    - the sample rate of the time-series to be processed
             olap  - overlap percentage, usually taken from the window function
             bmin  - the minimum bin number to be used. This is usually taken
                    from the window function.
             Lmin  - The minimum segment length.
             Jdes  - the desired number of frequencies.
             Kdes  - The desired number of averages.

OUTPUTS:     Each output is a vector, one value per frequency:
             f     - the frequency
             r     - frequency resolution (Hz)
             b     - bin number
             L     - segment lengths
             K     - number of averages

PARAMETER LIST:

M-FILE INFO: Get information about this methods by calling
             >> ao.getInfo('ltf_plan')

             Get information about a specified set-plist by calling:
             >> ao.getInfo('ltf_plan', 'None')

VERSION:     $Id: ltf_plan.m,v 1.4 2008/09/05 11:05:29 ingo Exp $

HISTORY:     19-02-2008 M Hewitson
             Creation

REFERENCE:   "lpsd revisited: ltf" / S2-AEI-TN-3052
             2008/02/07 V1.1
             G Heinzel
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [fs](#) FS Get the data property 'fs'.
-  [sqrt](#) SQRT computes the square root of the data in the AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.



SUBFUNCTIONS

 function ii = getInfo(varargin)
 function plo = getDefaultPlist()

SOURCE CODE [^]

```

0001 % LTF_PLAN computes all input values needed for the LPSD and LTFE algorithms.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LTF_PLAN computes all input values needed for the LPSD and LTFE
0005 %               algorithms.
0006 %
0007 % CALL:         [f, r, b, L, K] = ltf_plan(Ndata, fs, olap, bmin, Lmin, Jdes, Kdes)
0008 %
0009 % INPUTS:       Ndata - the length of the time-series to be processed
0010 %               fs   - the sample rate of the time-series to be processed
0011 %               olap  - overlap percentage, usually taken from the window function
0012 %               bmin  - the minimum bin number to be used. This is usually taken
0013 %                       from the window function.
0014 %               Lmin  - The minimum segment length.
0015 %               Jdes  - the desired number of frequencies.
0016 %               Kdes  - The desired number of averages.
0017 %
0018 % OUTPUTS:      Each output is a vector, one value per frequency:
0019 %               f     - the frequency
0020 %               r     - frequency resolution (Hz)
0021 %               b     - bin number
0022 %               L     - segment lengths
0023 %               K     - number of averages
0024 %
0025 % PARAMETER LIST:
0026 %
0027 % M-FILE INFO:  Get information about this methods by calling
0028 %               >> ao.getInfo('ltf_plan')
0029 %
0030 %               Get information about a specified set-plist by calling:
0031 %               >> ao.getInfo('ltf_plan', 'None')
0032 %
0033 % VERSION:      $Id: ltf_plan.m,v 1.4 2008/09/05 11:05:29 ingo Exp $
0034 %
0035 % HISTORY:      19-02-2008 M Hewitson
0036 %               Creation
0037 %
0038 % REFERENCE:    "lpsd revisited: ltf" / S2-AEI-TN-3052
0039 %               2008/02/07 V1.1
0040 %               G Heinzel
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043 %
0044 %               q     - a vector of start indices for the segments
0045 %
0046 function varargout = ltf_plan(varargin)
0047
0048 % Check if this is a call for parameters
0049 if utils.helper.isinfocall(varargin{:})
0050     varargout{1} = getInfo(varargin{3});
0051     return
0052 end
0053
0054 %% ----- Check inputs -----
0055 if nargin ~= 7 || nargout ~= 5
0056     help(mfilename)
0057     error('### Incorrect usage');
0058 end
0059
0060 Ndata = varargin{1};
0061 fs    = varargin{2};
0062 olap  = varargin{3};
0063 bmin  = varargin{4};
0064 Lmin  = varargin{5};
0065 Jdes  = varargin{6};
0066 Kdes  = varargin{7};
0067
0068 %% ----- Set up some variables -----
0069
0070 xov    = (1 - olap/100);
0071 fmin   = fs / Ndata * bmin;
0072 fmax   = fs/2;
0073 fresmin = fs / Ndata;
0074 freslim = fresmin * (1+xov*(Kdes-1));

```

```

0075     logfact = (Ndata/2)^(1/Jdes) - 1;
0076
0077
0078
0079     %% ----- Prepare outputs -----
0080
0081     f = [];
0082     r = [];
0083     b = [];
0084     L = [];
0085     K = [];
0086     % q = [];
0087
0088     %% ----- Loop over frequency -----
0089     fi = fmin;
0090     while fi < fmax
0091
0092         fres = fi * logfact;
0093         if fres <= freslim
0094             fres = sqrt(fres*freslim);
0095         end
0096         if fres < fresmin
0097             fres = fresmin;
0098         end
0099
0100         bin = fi/fres;
0101         if bin < bmin
0102             bin = bmin;
0103             fres = fi/bin;
0104         end
0105
0106         dftlen = round(fs / fres);
0107         if dftlen > Ndata
0108             dftlen = Ndata;
0109         end
0110         if dftlen < Lmin
0111             dftlen = Lmin;
0112         end
0113
0114         nseg = round((Ndata - dftlen) / (xov*dftlen) + 1);
0115         if nseg == 1
0116             dftlen = Ndata;
0117         end
0118
0119         fres = fs / dftlen;
0120         bin = fi / fres;
0121
0122         % Store outputs
0123         f = [f fi];
0124         r = [r fres];
0125         b = [b bin];
0126         L = [L dftlen];
0127         K = [K nseg];
0128
0129         fi = fi + fres;
0130
0131     end
0132
0133     %% ----- Set outputs -----
0134
0135     varargout{1} = f.';
0136     varargout{2} = r.';
0137     varargout{3} = b.';
0138     varargout{4} = L.';
0139     varargout{5} = K.';
0140 end
0141
0142 %-----
0143 % Get Info Object
0144 %-----
0145 function ii = getInfo(varargin)
0146     if nargin == 1 && strcmpi(varargin{1}, 'None')
0147         sets = {};
0148         pl = [];
0149     else
0150         sets = {'Default'};
0151         pl = getDefaultPlist();
0152     end
0153     % Build info object
0154     ii = minfo(mfilename, 'ao', '', utils.const.categories.relop, '$Id: ltf_plan.m,v 1.4
2008/09/05 11:05:29 ingo Exp $', sets, pl);
0155 end
0156
0157 %-----
0158 % Get Default Plist
0159 %-----

```

```
0160 function plo = getDefaultPlist()
0161
0162     plo = plist(...
0163         'Ndata', 10, ...
0164         'fs', 10, ...
0165         'olap', 50, ...
0166         'bmin', 1, ...
0167         'Lmin', 1e20, ...
0168         'Jdes', 1000, ...
0169         'Kdes', 100 ...
0170     );
0171 end
0172
0173
```

lxspecc

PURPOSE ^

LXSPEC performs log-scale cross-spectral analysis of various forms.

SYNOPSIS ^

function varargout = lxspecc(varargin)

DESCRIPTION ^

LXSPEC performs log-scale cross-spectral analysis of various forms.
%%%

DESCRIPTION: LXSPEC performs log-scale cross-spectral analysis of various forms.
The function is a helper function for various higher level
functions. It is meant to be called from other functions
(e.g., ltfe).

CALL: b = lxspecc(a, pl, method, iALGO, iVER, invars);

INPUTS: a - vector of input AOs
 pl - input parameter list
 method - one of
 'cpsd' - compute cross-spectral density
 'tfe' - estimate transfer function between inputs
 'cohere' - estimate cross-coherence
 mi - minfo object for calling method
 invars - invars variable from the calling higher level script

OUTPUTS: b - Na x Na matrix of output AOs

VERSION: \$Id: lxspecc.m,v 1.10 2008/09/03 17:05:51 ingo Exp \$

HISTORY: 16-05-2008 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [len](#) LEN overloads the length operator for Analysis objects. Length of the data samples.
-  [resample](#) RESAMPLE overloads resample function for AOs.
-  [select](#) SELECT select particular samples from the input AOs and return new AOs with only those samples.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % LXSPEC performs log-scale cross-spectral analysis of various forms.
```

```

0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LXSPEC performs log-scale cross-spectral analysis of various forms.
0005 %               The function is a helper function for various higher level
0006 %               functions. It is meant to be called from other functions
0007 %               (e.g., ltfe).
0008 %
0009 % CALL:         b = lxspecc(a, pl, method, iALGO, iVER, invars);
0010 %
0011 % INPUTS:       a      - vector of input AOs
0012 %               pl     - input parameter list
0013 %               method - one of
0014 %                   'cpsd'   - compute cross-spectral density
0015 %                   'tfe'    - estimate transfer function between inputs
0016 %                   'cohere' - estimate cross-coherence
0017 %               mi      - minfo object for calling method
0018 %               invars  - invars variable from the calling higher level script
0019 %
0020 % OUTPUTS:      b      - Na x Na matrix of output AOs
0021 %
0022 % VERSION:      $Id: lxspecc.m,v 1.10 2008/09/03 17:05:51 ingo Exp $
0023 %
0024 % HISTORY:      16-05-2008 M Hewitson
0025 %               Creation
0026 %
0027 %
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %
0030 function varargout = lxspecc(varargin)
0031
0032     import utils.const.*
0033
0034     % unpack inputs
0035     as      = varargin{1};
0036     pl      = varargin{2};
0037     method  = varargin{3};
0038     mi      = varargin{4};
0039     invars  = varargin{5};
0040
0041     VERSION = '$Id: lxspecc.m,v 1.10 2008/09/03 17:05:51 ingo Exp $';
0042     % Set the method version string in the minfo object
0043     mi.setMversion([VERSION '-->' mi.mversion]);
0044
0045     na      = numel(as);
0046     copies  = zeros(size(as));
0047
0048     %----- Resample all AOs
0049     fsmax = ao.findFsMax(as);
0050     fspl  = plist(param('fsout', fsmax));
0051     for i=1:na
0052         % check this is a time-series object
0053         if ~isa(as(i).data, 'tsdata')
0054             warning('### xspecc requires tsdata (time-series) inputs. Skipping AO %s',
invars{i});
0055             as(i) = [];
0056         else
0057             % Check Fs
0058             if as(i).data.fs ~= fsmax
0059                 utils.helper.msg(msg.PROC2, 'resampling AO %s to %f Hz', as(i).name, fsmax);
0060                 % Make a deep copy so we don't
0061                 % affect the original input data
0062                 as(i) = copy(as(i), 1);
0063                 copies(i) = 1;
0064                 as(i) = resample(as(i), fspl);
0065             end
0066         end
0067     end
0068
0069     %----- Truncate all vectors
0070     % Get shortest vector
0071     lmin = ao.findShortestVector(as);
0072     nsecs = lmin / fsmax;
0073     for i=1:na
0074         if len(as(i)) ~= lmin
0075             utils.helper.msg(msg.PROC1, 'truncating AO %s to %d secs', as(i).name, nsecs);
0076             % do we already have a copy?
0077             if ~copies(i)
0078                 % Make a deep copy so we don't
0079                 % affect the original input data
0080                 as(i) = copy(as(i), 1);
0081                 copies(i) = 1;
0082             end
0083             as(i) = select(as(i), 1:lmin);
0084         end
0085     end
0086

```

```

0087 %----- Build signal Matrix
0088 N      = len(as(1)); % length of first signal
0089 iS     = zeros(na, N);
0090 for j=1:na
0091     iS(j,:) = as(j).data.getY;
0092 end
0093
0094 %----- check input parameters
0095
0096 % Averaging
0097 Kdes = find(pl, 'Kdes'); % desired averages
0098 Jdes = find(pl, 'Jdes'); % num desired spectral frequencies
0099 Win  = find(pl, 'Win');  % Window object
0100 Nlap = find(pl, 'Olap'); % desired overlap
0101 if Nlap == -1
0102     % use window function rov
0103     Nlap = Win.rov/100;
0104     utils.helper.msg(msg.PROC1, 'using recommended overlap of %d', Nlap);
0105     pl = pset(pl, 'Olap', Nlap);
0106 end
0107 Order = find(pl, 'Order'); % desired overlap
0108 Lmin = find(pl, 'Lmin'); % Minimum segment length
0109
0110 %----- Get frequency vector
0111 [f, r, m, L, K] = ao.ltf_plan(Lmin, fsmax, Nlap, 1, Lmin, Jdes, Kdes);
0112
0113 %----- compute TF Estimates
0114 Txy = ao.mltfe(iS, f, r, m, L, fsmax, Win, Order, Nlap*100, Lmin, method);
0115
0116 %----- Build output Matrix of AOs
0117 utils.helper.msg(msg.PROC1, 'building output AO matrix');
0118 bs(na,na) = ao;
0119 for j=1:na % Loop over input signal
0120     for k=1:na % Loop over output signal
0121         % Keep the data shape of the first AO
0122         y = squeeze(Txy(j,k,:));
0123         if size(as(1).data.y,1) == 1
0124             f = f.';
0125             y = y.';
0126         end
0127         % create new output fsdata
0128         fsd = fsdata(f, y, fsmax);
0129         fsd.setYunits(as(k).data.yunits / as(j).data.yunits);
0130         % make output analysis object
0131         bs(j,k) = ao(fsd);
0132         % add history
0133         bs(j,k).addHistory(mi, pl, [invars(j) invars(k)], [as(j).hist, as(k).hist]);
0134         % set name
0135         bs(j,k).setName(sprintf('%s(%s->%s)', method, invars{j}, invars{k}), 'internal');
0136     end % End loop over output signal
0137 end % End loop over input signal
0138
0139 % Set output
0140 varargout{1} = bs;
0141 end
0142 % END

```

mchol

PURPOSE [^]

MCHOL

SYNOPSIS [^]

function varargout = mchol(m)

DESCRIPTION [^]

MCHOL

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [det](#) DET overloads the determinant function for Analysis objects.
-  [sqrt](#) SQRT computes the square root of the data in the AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE [^]

```
0001 % MCHOL
0002 function varargout = mchol(m)
0003
0004     leng = length(m);
0005     for i=1:leng
0006         dets(i) = det(m(1:i, 1:i));
0007     end
0008
0009     sqdet = sqrt(dets);
0010     if isreal('sqdet')==0
0011         error('### matrix not positive definite')
0012     end
0013
0014     if(isa(m, 'sym'))
0015         M = m;
0016         n = length( M );
0017         L = vpa(zeros( n, n ));
0018         for i=1:n
0019             L(i, i) = sqrt( M(i, i) - L(i, :)*L(i, :)' );
0020             for j=(i + 1):n
0021                 L(j, i) = ( M(j, i) - L(i, :)*L(j, :)' )/L(i, i);
0022             end
0023         end
0024         % L = solve('L');
0025         varargout{1} = L;
0026     end
0027
0028 end
```


mdc1_input_noises

PURPOSE ^

LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.

SYNOPSIS ^

function varargout = mdc1_input_noises(varargin)

DESCRIPTION ^

```
LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO
             sensing noise for MDC1.
CALL:       b = ltpda_mdc1_ifo_noise(pl)
PARAMETERS:
             'source' - Choose source of noise [default: 'ifo']
                    'ifo' - IFO sensing noise [m^2/Hz]
                    'sc'  - Spacecraft force noise [m^4s^{-2}/Hz]
                    'tm'  - Test-mass force noise [m^4s^{-2}/Hz]
             'f'      - a vector of frequencies [default: 1]
or
             'f1'     - start frequency [default: 1e-6]
             'f2'     - stop frequency  [default: 5]
             'nf'     - number of frequency points [default: 1000]
             'scale'  - frequency spacing, 'lin' or 'log' [default: 'log']
VERSION:       $Id: mdc1_input_noises.m,v 1.3 2008/09/05 11:05:29 ingo Exp $
The following call returns a parameter list object that contains the
default parameter values:
>> pl = ltpda_mdc1_ifo_noise(ao, 'Params')
The following call returns a string that contains the routine CVS version:
>> version = ltpda_mdc1_ifo_noise(ao, 'Version')
The following call returns a string that contains the routine category:
>> category = ltpda_mdc1_ifo_noise(ao, 'Category')
HISTORY: 11-04-08 M Hewitson
          Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Check if this is a call for parameters
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ao](#) AO analysis object class constructor.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [log10](#) LOG10 overloads the log10 operator for Analysis objects. Common (base 10) logarithm.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS [^]

-  function o = getIfoNoise(f)
-  function o = getSCNoise(f)
-  function o = getTMNoise(f)
-  function plo = getDefaultPlist(varargin)
-  function ii = getInfo(varargin)

SOURCE CODE [^]

```

0001 % LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO
0006 %               sensing noise for MDC1.
0007 %
0008 % CALL:         b = ltpda_mdc1_ifo_noise(pl)
0009 %
0010 % PARAMETERS:
0011 %
0012 %             'source' - Choose source of noise [default: 'ifo']
0013 %                       'ifo' - IFO sensing noise [m^2/Hz]
0014 %                       'sc'  - Spacecraft force noise [m^4s^{-2}/Hz]
0015 %                       'tm'  - Test-mass force noise [m^4s^{-2}/Hz]
0016 %             'f'      - a vector of frequencies [default: 1]
0017 %   or
0018 %             'f1'      - start frequency [default: 1e-6]
0019 %             'f2'      - stop frequency  [default: 5]
0020 %             'nf'      - number of frequency points [default: 1000]
0021 %             'scale'   - frequency spacing, 'lin' or 'log' [default: 'log']
0022 %
0023 % VERSION:       $Id: mdc1_input_noises.m,v 1.3 2008/09/05 11:05:29 ingo Exp $
0024 %
0025 % The following call returns a parameter list object that contains the
0026 % default parameter values:
0027 %
0028 % >> pl = ltpda_mdc1_ifo_noise(ao, 'Params')
0029 %
0030 % The following call returns a string that contains the routine CVS version:
0031 %
0032 % >> version = ltpda_mdc1_ifo_noise(ao, 'Version')
0033 %
0034 % The following call returns a string that contains the routine category:
0035 %
0036 % >> category = ltpda_mdc1_ifo_noise(ao, 'Category')
0037 %
0038 % HISTORY: 11-04-08 M Hewitson
0039 %           Creation
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 function varargout = mdc1_input_noises(varargin)
0043 %%% Check if this is a call for parameters
0044 if utils.helper.isinfocall(varargin{:})
0045     varargout{1} = getInfo(varargin{3});
0046     return
0047 end
0048
0049 %%% Collect input variable names
0050 in_names = cell(size(varargin));
0051 for ii = 1:nargin, in_names{ii} = inputname(ii); end
0052
0053 pli      = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0054
0055 %%% Decide on a deep copy or a modify
0056 %%% REMARK: If you create a new AO (call the constructor) then
0057 %%%         it is not necessary to copy the input-AOs !!!!!!!!!!!!!!!!!!!!!!!
0058
0059
0060 %%% Combine plists
0061 pl = combine(pli, getDefaultPlist('Range'));
0062
0063
0064
0065 %%% Extract parameters from plist
0066
0067 f = find(pl, 'f');
0068 if isempty(f)
0069     f1 = find(pl, 'f1');

```

```

0070     f2 = find(pl, 'f2');
0071     nf = find(pl, 'nf');
0072     scale = find(pl, 'scale');
0073
0074     switch scale
0075     case 'lin'
0076         f = linspace(f1, f2, nf);
0077     case 'log'
0078         f = logspace(log10(f1), log10(f2), nf);
0079     end
0080 end
0081
0082 source = find(pl, 'source');
0083
0084 %% Compute response for frequencies f
0085
0086 switch source
0087 case 'ifo'
0088     a = getIfoNoise(f);
0089 case 'tm'
0090     a = getTMNoise(f);
0091 case 'sc'
0092     a = getSCNoise(f);
0093 otherwise
0094     error('### Unknown noise source requested.');
```

```

0095 end
0096
0097 varargout{1} = a;
0098 end
0099 %-----
0100 % Get IFO noise for each frequency
0101 function o = getIfoNoise(f)
0102
0103     % Description of the noise
0104     fcn = '0.5*(5e-12)^2 * ( 1 + (1+1e8)./(1+f.^2/1e-12) .* (1 + (1+9e6)./(1+f.^2/1e-14) ) )';
0105
0106     % Pack in a plist
0107     pl = plist('fsfcn', fcn, 'f', f);
0108
0109     % Build A0 from plist
0110     o = ao(pl);
0111     o.setName('o_nxx', 'internal');
0112     o.setXunits('Hz', 'internal');
0113     o.setYunits('m^2/Hz', 'internal');
```

```

0114 end
0115 %-----
0116 % Get SC noise for each frequency
0117 function o = getSCNoise(f)
0118
0119     % Zeros as constant factors
0120     z1 = 1 + (10e-3/1e-6)^2;
0121     z2 = 1 + (0.1e-3/0.1e-6)^2;
0122     % AOs for frequency-dependent parts
0123     p1 = ao(plist('fsfcn', '1 + (f./100).^2', 'f', f));
0124     p2 = ao(plist('fsfcn', '1 + (f/1e-6).^2', 'f', f));
0125     p3 = ao(plist('fsfcn', '1 + (f/0.1e-6).^2', 'f', f));
0126
0127     % Combine the parts
0128     G = (0.1e-6/436)^2;
0129     o = G .* (1./p1 + (z1./p2).*(1 + z2./p3 ));
0130
0131     % Set properties of final AO
0132     o.setName('A_nxx', 'internal');
0133     o.setXunits('Hz', 'internal');
0134     o.setYunits('m^2s^4/ Hz', 'internal');
```

```

0135 end
0136 %-----
0137 % Get TM noise for each frequency
0138 function o = getTMNoise(f)
0139
0140     % Construct the various parts of the spectrum
0141
0142     % zeros
0143     z1 = ao(plist('fsfcn', '(1 + (f./100).^2)', 'f', f));
0144     z2 = (1 + (3e-3/1e-6).^2);
0145     z3 = ao(plist('fsfcn', '(1 + (f./1e-6).^2)', 'f', f));
0146     z4 = (1 + (0.1e-3/0.1e-6).^2);
0147     z5 = ao(plist('fsfcn', '(1 + (f./0.1e-6).^2)', 'f', f));
0148
0149     % poles
0150     p1 = (1 + (1e-3./100).^2);
0151     p2 = (1 + (3e-3./1e-6).^2);
0152     p3 = (1 + (1e-3./1e-6).^2);
0153     p4 = (1 + (0.1e-3./0.1e-6).^2);
0154     p5 = (1 + (1e-3./0.1e-6).^2);
```

```

0155
0156 % Combine the parts
0157 o = (30e-15)^2 .* ( (1./z1 + z2./z3.*(1 + z4./z5)) ./ (1./p1 + p2./p3.*(1 +
p4./p5)));
0158
0159 % Set properties of final AO
0160 o.setName('A_1xx', 'internal');
0161 o.setXunits('Hz', 'internal');
0162 o.setYunits('m^2s^[-4]/ Hz', 'internal');
0163
0164 end
0165 %% Default parameters
0166 function plo = getDefaultPlist(varargin)
0167 % List of available parameter sets
0168 sets = {'List', 'Range'};
0169 if nargin == 0
0170     plo = sets;
0171     return
0172 end
0173 set = varargin{1};
0174 switch set
0175     case 'List'
0176         plo = plist('source', 'ifo', 'f', [1]);
0177     case 'Range'
0178         plo = plist('source', 'ifo', ...
0179                     'f1', 1e-6, ...
0180                     'f2', 5, ...
0181                     'nf', 1000, ...
0182                     'scale', 'log');
0183     otherwise
0184         plo = plist();
0185 end
0186 end
0187
0188 %-----
0189 function ii = getInfo(varargin)
0190 if nargin == 1 && strcmpi(varargin{1}, 'None')
0191     sets = {};
0192     pl = [];
0193 else
0194     sets = {'Default'};
0195     pl = getDefaultPlist;
0196 end
0197 % Build info object
0198 ii = minfo(mfilename, 'ao', '', utils.const.categories.mdc01, '$Id:
mdc1_input_noises.m,v 1.3 2008/09/05 11:05:29 ingo Exp $', sets, pl);
0199 end

```

[Home](#) > [classes](#) > [@ao](#) > mlpsd_m.m

mlpsd_m

PURPOSE ^

MLPSD_M m-file only version of the LPSD algorithm

SYNOPSIS ^

function varargout = mlpsd_m(varargin)

DESCRIPTION ^

```
MLPSD_M m-file only version of the LPSD algorithm
[S,Sxx,ENBW] = mlpsd_m(x,f,r,m,L,fs,win,order,olap)
M Hewitson 19-05-07
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [conj](#) CONJ overloads the conjugate operator for Analysis objects.
-  [detrend](#) DETREND detrends the input analysis object using a polynomial of degree N.
-  [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
-  [fs](#) FS Get the data property 'fs'.
-  [mean](#) MEAN computes the mean value of the data in the AO.
-  [x](#) X Get the data property 'x'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % MLPSD_M m-file only version of the LPSD algorithm
0002 %
0003 % [S,Sxx,ENBW] = mlpsd_m(x,f,r,m,L,fs,win,order,olap)
0004 %
0005 % M Hewitson 19-05-07
0006 %
0007 %
0008
0009 function varargout = mlpsd_m(varargin)
0010
0011     import utils.const.*
0012
0013     utils.helper.msg(msg.PROC1, 'using MATLAB to compute core DFT');
0014
0015     % Get inputs
0016     x = varargin{1};
0017     f = varargin{2};
0018     r = varargin{3};
0019     m = varargin{4};
0020     L = varargin{5};
0021     fs = varargin{6};
0022     win = varargin{7};
0023     order = varargin{8};
0024     olap = varargin{9};
0025
0026     twopi = 2.0*pi;
0027
0028     nx = length(x);
```

```

0029 nf = length(f);
0030 Sxx = zeros(nf,1);
0031 S = zeros(nf,1);
0032 ENBW = zeros(nf,1);
0033
0034 disp_each = round(nf/100)*10;
0035
0036 for j=1:nf
0037
0038     if mod(j,disp_each) == 0 || j == 1 || j == nf
0039         utils.helper.msg(msg.PROC1, 'computing frequency %04d of %d: %f Hz', j, nf, f(j));
0040     end
0041
0042     % compute DFT exponent and window
0043     l = L(j);
0044     switch win.type
0045     case 'Kaiser'
0046         win = specwin(win.type, l, win.psl1);
0047     otherwise
0048         win = specwin(win.type, l);
0049     end
0050
0051     p = 1i * twopi * m(j)/l.*[0:l-1];
0052     C = win.win .* exp(p);
0053     Xolap = (1-olap);
0054     % do segments
0055     A = 0.0;
0056
0057     % Compute the number of averages we want here
0058     segLen = l;
0059     nData = length(x);
0060     ovfact = 1 / (1 - olap);
0061     davg = (((nData - segLen)) * ovfact) / segLen + 1;
0062     navg = round(davg);
0063
0064     % Compute steps between segments
0065     if navg == 1
0066         shift = 1;
0067     else
0068         shift = (nData - segLen) / (navg - 1);
0069     end
0070     if shift < 1 || isnan(shift)
0071         shift = 1;
0072     end
0073
0074     % disp(sprintf('Seglen: %d\t | Shift: %f\t | navs: %d', segLen, shift, navg))
0075
0076     start = 1.0;
0077     for ii = 1:navg
0078         % compute start index
0079         istart = round(start);
0080         start = start + shift;
0081
0082         % get segment
0083         xs = [x(istart:istart+l-1)].';
0084
0085         % detrend segment
0086         switch order
0087         case -1
0088             % do nothing
0089         case 0
0090             xs = xs - mean(xs);
0091         case 1
0092             xs = detrend(xs);
0093         otherwise
0094             xs = polydetrend(xs, order);
0095         end
0096
0097         % make DFT
0098         a = C*xs;
0099         A = A + a*conj(a);
0100
0101     end
0102
0103     if mod(j,disp_each) == 0 || j == 1 || j == nf
0104         utils.helper.msg(msg.PROC2, 'averaged %d segments', navg);
0105     end
0106
0107     A2ns = 2.0*A/navg;
0108     S1 = win.ws;
0109     S12 = S1*S1;
0110     S2 = win.ws2;
0111     ENBW(j) = fs*S2/S12;
0112     Sxx(j) = A2ns/fs/S2;
0113     S(j) = A2ns/S12;
0114

```

```
0115     end % for j=1:nf
0116
0117     varargout{1} = S;
0118     varargout{2} = Sxx;
0119     varargout{3} = ENBW;
0120
0121 end
0122
```

[Home](#) > [classes](#) > [@ao](#) > mlpsd_mex.m

mlpsd_mex

PURPOSE [^]

MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD

SYNOPSIS [^]

function varargout = mlpsd_mex(varargin)

DESCRIPTION [^]

MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD algorithm

[S,Sxx,ENBW] = mlpsd_mex(x,f,r,m,L,fs,win,order,olap)

M Hewitson 19-05-07

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.

 [fs](#) FS Get the data property 'fs'.

 [x](#) X Get the data property 'x'.

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE [^]

```

0001 % MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD
0002 % algorithm
0003 %
0004 % [S,Sxx,ENBW] = mlpsd_mex(x,f,r,m,L,fs,win,order,olap)
0005 %
0006 % M Hewitson 19-05-07
0007 %
0008 %
0009
0010 function varargout = mlpsd_mex(varargin)
0011
0012     import utils.const.*
0013
0014     utils.helper.msg(msg.PROC1, 'using ltpda_dft.mex to compute core DFT');
0015
0016     % Get inputs
0017     x      = varargin{1};
0018     f      = varargin{2};
0019     r      = varargin{3};
0020     m      = varargin{4};
0021     L      = varargin{5};
0022     fs     = varargin{6};
0023     win    = varargin{7};
0024     order  = varargin{8};
0025     olap   = varargin{9};
0026     Lmin   = varargin{10};
0027
0028     twopi  = 2.0*pi;
0029
0030     nx     = length(x);
0031     nf     = length(f);
0032     Sxx    = zeros(nf,1);

```



```

0033 S = zeros(nf,1);
0034 ENBW = zeros(nf,1);
0035
0036 disp_each = round(nf/100)*10;
0037
0038 minReached = 0;
0039
0040 for j=1:nf
0041
0042     if mod(j,disp_each) == 0 || j == 1 || j == nf
0043         utils.helper.msg(msg.PROC1, 'computing frequency %04d of %d: %f Hz', j, nf, f(j));
0044     end
0045
0046     % compute DFT exponent and window
0047     l = L(j);
0048
0049     % Check if we need to update the window values
0050     % - once we reach Lmin, the window values don't change.
0051     if ~minReached
0052         switch win.type
0053             case 'Kaiser'
0054                 win = specwin(win.type, l, win.psl1);
0055             otherwise
0056                 win = specwin(win.type, l);
0057         end
0058         if l==Lmin
0059             minReached = 1;
0060         end
0061     end
0062
0063     p = 1i * twopi * m(j)/l.*(0:l-1);
0064     C = win.win .* exp(p);
0065
0066     % Core DFT part in C-mex file
0067     [A, nsegs] = ltpda_dft(x, l, C, olap, order);
0068
0069     if mod(j,disp_each) == 0 || j == 1 || j == nf
0070         utils.helper.msg(msg.PROC2, 'averaged %d segments', nsegs);
0071     end
0072
0073     A2ns = 2.0*A/nsegs;
0074     S1 = win.ws;
0075     S12 = S1*S1;
0076     S2 = win.ws2;
0077     ENBW(j) = fs*S2/S12;
0078     Sxx(j) = A2ns/fs/S2;
0079     S(j) = A2ns/S12;
0080
0081 end
0082
0083 varargout{1} = S;
0084 varargout{2} = Sxx;
0085 varargout{3} = ENBW;
0086
0087 end
0088

```

[Home](#) > [classes](#) > [@ao](#) > mltfe.m

mltfe

PURPOSE ^

MLTFE compute log-frequency space TF

SYNOPSIS ^

function varargout = mltfe(varargin)

DESCRIPTION ^

```
MLTFE compute log-frequency space TF
Txy = mltfe(X,f,r,m,L,fs,win,order,olap,Lmin,method)
M Hewitson 19-05-07
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [exp](#) EXP overloads the exp operator for Analysis objects. Exponential.
-  [fs](#) FS Get the data property 'fs'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```

0001 % MLTFE compute log-frequency space TF
0002 %
0003 % Txy = mltfe(X,f,r,m,L,fs,win,order,olap,Lmin,method)
0004 %
0005 % M Hewitson 19-05-07
0006 %
0007 %
0008
0009 function varargout = mltfe(varargin)
0010
0011     import utils.const.*
0012
0013     % Get inputs
0014     X      = varargin{1};
0015     f      = varargin{2};
0016     r      = varargin{3};
0017     m      = varargin{4};
0018     L      = varargin{5};
0019     fs     = varargin{6};
0020     win    = varargin{7};
0021     order  = varargin{8};
0022     olap   = varargin{9};
0023     Lmin   = varargin{10};
0024     method = varargin{11};
0025
0026
0027     % --- Prepare some variables
0028     twopi  = 2.0*pi;
0029     si     = size(X);
0030     nc     = si(1);
0031     nf     = length(f);
0032     Txy    = zeros(nc,nc,nf);
0033     disp each = round(nf/100)*10;

```

```

0034 minReached = 0;
0035
0036 % ----- Loop over Frequency
0037 for fi=1:nf
0038
0039     % compute DFT exponent and window
0040     l = L(fi);
0041
0042     % Check if we need to update the window values
0043     % - once we reach Lmin, the window values don't change.
0044     if ~minReached
0045         switch win.type
0046             case 'Kaiser'
0047                 win = specwin(win.type, l, win.psl1);
0048             otherwise
0049                 win = specwin(win.type, l);
0050         end
0051         if l==Lmin
0052             minReached = 1;
0053         end
0054     end
0055
0056     % Compute DFT coefficients
0057     p = 1i * twopi * m(fi)/l.*[0:l-1];
0058     C = win.win .* exp(p);
0059
0060     if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
0061         utils.helper.msg(msg.PROC1, 'computing frequency %04d of %04d: %f Hz', fi, nf,
0062 f(fi));
0063     end
0064
0065     % Loop over input channels
0066     for j=1:nc
0067         % loop over output channels
0068         for k=1:nc
0069             switch method
0070                 case 'tfe'
0071                     if k~=j
0072                         % Core cross-DFT part in C-mex file
0073                         % We need cross-spectrum and Power spectrum
0074                         [XY, XX, YY, nsegs] = ltpda_dft(X(j,:), X(k,:), l, C, olap, order);
0075                         Txy(j,k,fi) = XY/XX;
0076                     else
0077                         % Trivial case of equal input and output
0078                         Txy(j,k,fi) = 1;
0079                     end
0080                 case 'cpsd'
0081                     if k~=j
0082                         % Core cross-DFT part in C-mex file
0083                         [A, XX, YY, nsegs] = ltpda_dft(X(j,:), X(k,:), l, C, olap, order);
0084                     else
0085                         % Core DFT part in C-mex file
0086                         [A, nsegs] = ltpda_dft(X(j,:), l, C, olap, order);
0087                     end
0088                     S1 = win.ws;
0089                     S2 = win.ws2;
0090                     Txy(j,k,fi) = 2.0*A/nsegs/fs/S2;
0091                 case 'cohere'
0092                     if k>j
0093                         % Core cross-DFT part in C-mex file
0094                         % We need cross-spectrum and Power spectrum
0095                         [XY, XX, YY, nsegs] = ltpda_dft(X(j,:), X(k,:), l, C, olap, order);
0096                         Txy(j,k,fi) = (abs(XY).^2)/(XX.*YY);
0097                     else
0098                         % Trivial case of equal input and output
0099                         Txy(j,k,fi) = 1;
0100                     end
0101                 end
0102             end % End loop over output channels
0103         end % End loop over input channels
0104     end % End loop over frequencies
0105
0106     switch method
0107         case 'cohere'
0108             % Fill other elements
0109             for j=1:nc
0110                 for k=1:nc
0111                     if k<j
0112                         Txy(j,k,:) = Txy(k,j,:);
0113                     end
0114                 end
0115             end
0116         end
0117     end
0118

```

```
0119     case 'tfe'  
0120     case 'cpsd'  
0121 end  
0122  
0123 % Set output  
0124 varargout{1} = Txy;  
0125  
0126 end  
0127
```

ngconv

PURPOSE ^

NGCONV is called by the function LTPDA_NOISEGEN

SYNOPSIS ^

function [a,b] = ngconv(pzm)

DESCRIPTION ^

NGCONV is called by the function LTPDA_NOISEGEN
it takes the pole zero model pzm (user input to LTPDA_NOISEGEN)
as input and returns as
 outputs:
 - a: numerator coefficients
 - b: denominator coefficients
 of the target transfer function

A Monsky 24-07-07

\$Id: ngconv.m,v 1.3 2008/08/01 13:19:42 ingo Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % NGCONV is called by the function LTPDA_NOISEGEN
0002 % it takes the pole zero model pzm (user input to LTPDA_NOISEGEN)
0003 % as input and returns as
0004 %     outputs:
0005 %         - a: numerator coefficients
0006 %         - b: denominator coefficients
0007 %             of the target transfer function
0008 % A Monsky 24-07-07
0009 %
0010 % $Id: ngconv.m,v 1.3 2008/08/01 13:19:42 ingo Exp $
0011
0012 function [a,b] = ngconv(pzm)
0013
0014     zs = pzm.zeros;
0015     ps = pzm.poles;
0016
0017     f_zer = zeros(length(zs));
0018     q_zer = zeros(length(zs));
0019     for j=1:length(zs)
0020         z = zs(j);
0021         f_zer(j,1) = z.f;
0022         q_zer(j,1) = z.q;
0023         %if isnan(q_zer(j))
0024         %     q_zer(j,1) = 0;
0025         %end
0026         %zv(j,1:2) = [f q];
0027     end
0028
0029     f_pol = zeros(length(ps));
0030     q_pol = zeros(length(ps));
0031     for j=1:length(ps)
0032         p = ps(j);
0033         f_pol(j,1) = p.f;
0034         q_pol(j,1) = p.q;
```

```
0035     %if isnan(q_pol(j))
0036     %     q_pol(j,1) = 0;
0037     %end
0038 end
0039
0040 %%% calculate factors from f and q
0041 pol = ao.fq2fac(f_pol,q_pol);
0042 zer = ao.fq2fac(f_zer,q_zer);
0043
0044 %%%
0045 [b,a] = ao.conv_noisegen(pol,zer);
0046 end
0047
0048
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

nginit

PURPOSE ^

NGINIT is called by the function LTPDA_NOISEGEN

SYNOPSIS ^

function y = nginit(Tinit)

DESCRIPTION ^

NGINIT is called by the function LTPDA_NOISEGEN
it takes the matrix Tinit (calculated by NGSETUP) as input
and returns the initial state vector y

A Monsky 24-07-07

\$Id: nginit.m,v 1.2 2008/08/01 13:19:42 ingo Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

 [y](#) Y Get the data property 'y'.

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % NGINIT is called by the function LTPDA_NOISEGEN
0002 % it takes the matrix Tinit (calculated by NGSETUP) as input
0003 % and returns the initial state vector y
0004 %
0005 % A Monsky 24-07-07
0006 %
0007 % $Id: nginit.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0008
0009 function y = nginit(Tinit)
0010
0011     n = length(Tinit);
0012
0013     % writing the generator
0014     r = randn(n,1);
0015     y = Tinit * r;
0016
0017 end
0018
```


[Home](#) > [classes](#) > [@ao](#) > ngprop.m

ngprop

PURPOSE ^

NGPROP is called by the function LTPDA_NOISEGEN

SYNOPSIS ^

function [x y] = ngprop(Tprop, E, num, y, ns)

DESCRIPTION ^

NGPROP is called by the function LTPDA_NOISEGEN

Inputs calculated by ...

... NGCONV:
- num: numerator coefficients

... NGSETUP:
- Tprop: matrix to calculate propagation vector
- E: matrix to calculate propagation vector

... NGINIT
- y: initial state vector
- num: numerator coefficients

... USER
- ns: number of samples given as input from the user

Outputs:
- x: vector of timesamples
- y: last calculated state vector (could be used as input for next LTPDA_NOISEGEN call)

A Monsky 24-07-07

\$Id: ngprop.m,v 1.2 2008/08/01 13:19:42 ingo Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

 **x** X Get the data property 'x'.

 **y** Y Get the data property 'y'.

This function is called by:

 **ao** AO analysis object class constructor.

SOURCE CODE ^

```

0001 % NGPROP is called by the function LTPDA_NOISEGEN
0002 %
0003 % Inputs calculated by ...
0004 % ... NGCONV:
0005 % - num: numerator coefficients
0006 % ... NGSETUP:
0007 % - Tprop: matrix to calculate propagation vector
0008 % - E: matrix to calculate propagation vector
0009 % ... NGINIT
0010 % - y: initial state vector
0011 % - num: numerator coefficients
0012 % ... USER
0013 % - ns: number of samples given as input from the user
0014 % Outputs:
0015 % - x: vector of timesamples
0016 % - y: last calculated state vector (could be used as input
0017 % for next LTPDA_NOISEGEN call)
0018 % A Monsky 24-07-07
0019 %

```

```
0020 % $Id: ngprop.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0021
0022 function [x y] = ngprop(Tprop, E, num, y, ns)
0023
0024     lengT = length(Tprop);
0025     lengb = lengT+1;
0026
0027     num=num';
0028     num = [num zeros(1,(lengb-length(num)-1))];
0029
0030
0031     x = zeros(ns,1);
0032     for i=1:ns
0033         r = randn(lengT,1);
0034         y = E * y + Tprop * r;
0035         x(i) = num*y;
0036     end
0037
0038 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [classes](#) > [@ao](#) > ngsetup.m

ngsetup

PURPOSE [^]

NGSETUP is called by the function LTPDA_NOISEGEN

SYNOPSIS [^]

```
function [Tinit,Tprop,E] = ngsetup(den,fs)
```

DESCRIPTION [^]

```

NGSETUP is called by the function LTPDA_NOISEGEN
Inputs calculated by ...
... NGCONV:
    - den:    denominator coefficients

... USER
    - fs:     sampling frequency given as input to LTPDA_NOISEGEN

Outputs:
    - Tinit:  matrix to calculate initial state vector
    - Tprop:  matrix to calculate propagation vector
    - E:      matrix to calculate propagation vector

A Monsky 24-07-07
$Id: ngsetup.m,v 1.2 2008/08/01 13:19:42 ingo Exp $

```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [fs](#) FS Get the data property 'fs'.

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE [^]

```

0001 % NGSETUP is called by the function LTPDA_NOISEGEN
0002 %
0003 % Inputs calculated by ...
0004 % ... NGCONV:
0005 %     - den:    denominator coefficients
0006 %
0007 % ... USER
0008 %     - fs:     sampling frequency given as input to LTPDA_NOISEGEN
0009 %
0010 %     Outputs:
0011 %     - Tinit:  matrix to calculate initial state vector
0012 %     - Tprop:  matrix to calculate propagation vector
0013 %     - E:      matrix to calculate propagation vector
0014 %
0015 % A Monsky 24-07-07
0016 %
0017 % $Id: ngsetup.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0018
0019 function [Tinit,Tprop,E] = ngsetup(den,fs)
0020
0021     den=den';
0022     dt = 1/fs;
0023     length(den);
0024

```

```

0025 n = length(den)-1;
0026
0027 %% setting up matrix Aij
0028
0029 m_a = zeros(n,n);
0030 for i = 1:n
0031     for j = 1:n
0032         if j == i+1
0033             m_a(i,j) = 1;
0034         end
0035         if i == n
0036             m_a(i,j) = -den(j);
0037         end
0038     end
0039 end
0040
0041 %% Matrix exponential E
0042 E = expm(m_a*dt);
0043
0044 %% setting up matrix Bij
0045 B = zeros(n,n);
0046 for i=1:n
0047     if rem(i,2) ~= 0
0048         j0 = (i+1)/2;
0049         s = (-1)^(j0+1);
0050         j = j0;
0051         for k=1:2:(n+1)
0052             B(i,j) = s*den(k);
0053             s = -s;
0054             j = j+1;
0055         end
0056     end
0057     if rem(i,2) == 0
0058         j0 = i/2+1;
0059         s = (-1)^j0;
0060         j = j0;
0061         for k=2:2:(n+1)
0062             B(i,j) = s * den(k);
0063             s = -s;
0064             j = j+1;
0065         end
0066     end
0067 end
0068
0069 %% solve B * m = k
0070 m_k = zeros(n,1);
0071 m_k(n) = 0.5;
0072 m_m = B\m_k;
0073
0074 %% filling covariance matrix Cinit
0075 Cinit = zeros(n,n);
0076 for i=1:n
0077     for j=1:n
0078         if rem((i+j),2) == 0 % even
0079             Cinit(i,j) = (-1)^((i-j)/2) * m_m((i+j)/2);
0080         else
0081             Cinit(i,j) = 0;
0082         end
0083     end
0084 end
0085
0086
0087 %cholesky decomposition
0088
0089 Tinit = chol(Cinit,'lower'); %lower triangular matrix
0090
0091 %% setting up matrix D
0092 N = n*(n+1)/2;
0093
0094 m_d = zeros(N);
0095 g = zeros(n);
0096 for i=1:n
0097     for j=1:n
0098         if i>=j
0099             g(i,j) = (i*i-i)/2+(j);
0100         else
0101             g(i,j) = (j*j-j)/2+(i);
0102         end
0103     end
0104 end
0105
0106 for i=1:n
0107     for j=i:n
0108         for k=1:n
0109             m_d(g(i,j),g(j,k)) = m_d(g(i,j),g(j,k)) + m_a(i,k);
0110             m_d(g(i,j),g(i,k)) = m_d(g(i,j),g(i,k)) + m_a(j,k);

```

```

0111         end
0112     end
0113 end
0114
0115
0116 %% setting up q from  $D * p = q$ 
0117 m_q = zeros(1,g(n,n));
0118 for i=1:n
0119     for j=i:n
0120         if i==n
0121             m_q(g(i,j)) = (E(n,n))*(E(n,n))-1;
0122         else
0123             m_q(g(i,j)) = (E(i,n)*E(j,n));
0124         end
0125     end
0126 end
0127
0128 m_p = m_d\m_q';
0129
0130 Cprop = zeros(n);
0131 for i=1:n
0132     for j=1:n
0133         Cprop(i,j) = m_p(g(i,j));
0134     end
0135 end
0136 Tprop = chol(Cprop, 'lower');
0137
0138 % %% writing the generator
0139 % r = randn(n,1);
0140 % y = Tinit * r;
0141 % x = zeros(Nfft,1);
0142 % for i=1:Nfft
0143 %     r = randn(n,1);
0144 %     y = E * y + Tprop * r;
0145 %     x(i) = a*y;
0146 % end
0147
0148 end
0149

```

ngsetup_vpa

PURPOSE ^

ALGONAME = mfilename;

SYNOPSIS ^

function [Tinit,Tprop,E] = ngsetup_vpa(den,fs, ndigits)

DESCRIPTION ^

```
ALGONAME = mfilename;
VERSION   = '$Id: ngsetup_vpa.m,v 1.2 2008/08/01 13:19:42 ingo Exp $';
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [fs](#) FS Get the data property 'fs'.

This function is called by:

 [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 function [Tinit,Tprop,E] = ngsetup_vpa(den,fs, ndigits)
0002
0003     % ALGONAME = mfilename;
0004     % VERSION   = '$Id: ngsetup_vpa.m,v 1.2 2008/08/01 13:19:42 ingo Exp $';
0005     digits(ndigits)
0006     fs = sym(fs);
0007     den = sym(den, 'd');
0008     den=den';
0009     dt = vpa(1/fs);
0010
0011     n = length(den)-1;
0012     % digits(d);
0013
0014     %% setting up matrix Aij
0015     m_a = vpa(zeros(n,n));
0016     for i = 1:n
0017         for j = 1:n
0018             if j == i+1
0019                 m_a(i,j) = 1;
0020             end
0021             if i == n
0022                 m_a(i,j) = -den(j);
0023             end
0024         end
0025     end
0026
0027     %% Matrix exponential E
0028     a = m_a*dt;
0029     E = expm(a);
0030
0031
0032     %% setting up matrix Bij
0033     B = vpa(zeros(n,n));
0034     for i=1:n
0035         if rem(i,2) ~= 0
0036             j0 = (i+1)/2;
0037             s = (-1)^(j0+1);
0038             j = j0;
0039             for k=1:2:(n+1)
```

```

0040         d1 = den(k);
0041         d2 = s*d1;
0042         B(i,j) = d2;
0043         s = -s;
0044         j = j+1;
0045     end
0046 end
0047 if rem(i,2) == 0
0048     j0 = i/2+1;
0049     s = (-1)^j0;
0050     j = j0;
0051     for k=2:2:(n+1)
0052         d1 = den(k);
0053         d2 = s*d1;
0054         B(i,j) = d2;
0055         s = -s;
0056         j = j+1;
0057     end
0058 end
0059 end
0060
0061 %% solve B * m = k
0062 m_k = vpa(zeros(n,1));
0063 m_k(n) = 0.5;
0064 m_m = vpa(B\m_k);
0065
0066 %% filling covariance matrix Cinit
0067 % Cinit = vpa(zeros(n,n));
0068 % for i=1:n
0069 %     for j=1:n
0070 %         if rem((i+j),2) == 0 % even
0071 %             d1 = (-1)^((i-j)/2);
0072 %             d1 = subs(d1)
0073 %             d2 = vpa('m_m((i+j)/2)');
0074 %             d2 = subs(d2)
0075 %             d3 = vpa(ctranspose(d2));
0076 %             d3 = subs(d3)
0077 %             d4 = vpa('d1 * d3');
0078 %             d4 = subs(d4)
0079 %             Cinit(i,j) = d4;
0080 %         else
0081 %             Cinit(i,j) = 0;
0082 %         end
0083 %     end
0084 % end
0085
0086 Cinit = vpa(zeros(n,n));
0087 for i=1:n
0088     for j=1:n
0089         if rem((i+j),2) == 0 % even
0090             d1 = (-1)^((i-j)/2);
0091             d2 = (i+j)/2;
0092             Cinit(i,j) = d1 * m_m(d2);
0093             % else
0094             %     Cinit(i,j) = 0;
0095         end
0096     end
0097 end
0098 %cholesky decomposition
0099
0100 % Tinit = chol(double(Cinit),'lower'); %lower triangular matrix
0101 Tinit = ao.mchol(Cinit);
0102
0103 %% setting up matrix D
0104 N = n*(n+1)/2;
0105
0106 m_d = vpa(zeros(N));
0107 g = zeros(n);
0108 for i=1:n
0109     for j=1:n
0110         if i>=j
0111             g(i,j) = (i*i-i)/2+(j);
0112         else
0113             g(i,j) = (j*j-j)/2+(i);
0114         end
0115     end
0116 end
0117
0118 for i=1:n
0119     for j=i:n
0120         for k=1:n
0121             m_d(g(i,j),g(j,k)) = m_d(g(i,j),g(j,k)) + m_a(i,k);
0122             m_d(g(i,j),g(i,k)) = m_d(g(i,j),g(i,k)) + m_a(j,k);
0123         end
0124     end
0125 end

```

```

0126
0127
0128 %% setting up q from D * p = q
0129 m_q = vpa(zeros(1,g(n,n)));
0130 for i=1:n
0131     for j=i:n
0132         if i==n
0133             m_q(g(i,j)) = (E(n,n))*(E(n,n))-1;
0134         else
0135             m_q(g(i,j)) = (E(i,n)*E(j,n));
0136         end
0137     end
0138 end
0139
0140 m_p = m_d\m_q';
0141
0142 Cprop = vpa(zeros(n));
0143 for i=1:n
0144     for j=1:n
0145         Cprop(i,j) = m_p(g(i,j));
0146     end
0147 end
0148
0149 Tprop = ao.mchol(Cprop);
0150 Tprop = double(Tprop);
0151 E = double(E);
0152 Tinit = double(Tinit);
0153
0154 % Tprop = chol(Cprop,'lower');
0155
0156 % Tprop = chol(double((Cprop)),'lower');
0157 % Tprop = mchol(Cprop);
0158
0159 % %% writing the generator
0160 % r = randn(n,1);
0161 % y = Tinit * r;
0162 % x = zeros(Nfft,1);
0163 % for i=1:Nfft
0164 %     r = randn(n,1);
0165 %     y = E * y + Tprop * r;
0166 %     x(i) = a*y;
0167 % end
0168
0169 end
0170

```


t0

PURPOSE ^

T0 Get the data property 't0'.

SYNOPSIS ^

function varargout = t0(varargin)

DESCRIPTION ^

```
T0 Get the data property 't0'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the data property 't0'.

CALL:      val = ao.t0();

INPUTS:    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('t0')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('t0', 'None')

VERSION:    $Id: t0.m,v 1.1 2008/09/05 13:55:17 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [consolidate](#) CONSOLIDATE resamples all input AOs onto the same time grid.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fixfs](#) FIXFS resamples the input time-series to have a fixed sample rate.
-  [fromTSfcn](#) FROMTSFCN Construct an ao from a ts-function string
-  [fromWaveform](#) FROMWAVEFORM Construct an ao from a waveform
-  [interp](#) INTERP interpolate the values in the input AO(s) at new values.
-  [join](#) JOIN multiple AOs into a single AO.
-  [resample](#) RESAMPLE overloads resample function for AOs.
-  [select](#) SELECT select particular samples from the input AOs and return new AOs with only those samples.
-  [split](#) SPLIT split an analysis object into the specified segments.
-  [timeshift](#) TIMESHIFT for AO/tsdata objects, shifts the time axis such that x(1) = 0.

SUBFUNCTIONS ^

 function ii = getInfo(varargin)

 `function plo = getDefaultPlist()`

SOURCE CODE ^

```

0001 % T0 Get the data property 't0'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the data property 't0'.
0005 %
0006 % CALL:      val = ao.t0();
0007 %
0008 % INPUTS:    obj - must be a single data2D object.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %              >> ao.getInfo('t0')
0012 %
0013 %              Get information about a specified set-plist by calling:
0014 %              >> ao.getInfo('t0', 'None')
0015 %
0016 % VERSION:   $Id: t0.m,v 1.1 2008/09/05 13:55:17 ingo Exp $
0017 %
0018 % HISTORY:    27-05-2008 Diepholz
0019 %              Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = t0(varargin)
0024
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 obj = varargin{1};
0032
0033 if isprop(obj.data, 't0')
0034     varargout{1} = obj.data.t0;
0035 else
0036     varargout{1} = [];
0037 end
0038
0039 end
0040
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %                               Local Functions                               %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:    getInfo
0048 %
0049 % DESCRIPTION: Get Info Object
0050 %
0051 % HISTORY:     11-07-07 M Hewitson
0052 %              Creation.
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ao', '', utils.const.categories.internal, '$Id: t0.m,v 1.1
2008/09/05 13:55:17 ingo Exp $', sets, pl);
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:    getDefaultPlist
0071 %
0072 % DESCRIPTION: Get Default Plist
0073 %
0074 % HISTORY:     11-07-07 M Hewitson
0075 %              Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %

```

```
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.2 2008/08/01 13:19:42 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.2 2008/08/01 13:19:42 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020     obj_struct = varargin{1};
0021     struct_ver = varargin{2};
0022
0023     % get the version of the current toolbox
0024     tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026     % get only the version string without the MATLAB version
0027     struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.0' -> 1.1' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034
0035         obj_struct.creator = obj_struct.provenance;
0036         obj_struct = rmfield(obj_struct, 'provenance');
0037
0038         obj_struct = rmfield(obj_struct, 'plist');
0039
0040         obj_struct.procinfo = plist();
0041         obj_struct.plotinfo = plist();
0042
0043     catch ME
0044         disp(varargin{1});
0045         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (ao) above is not
from version 1.0')));
0046     end
0047
0048     % Set the struct_ver to '1.1'
0049     struct_ver = '1.1';
0050
0051 end
0052
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054
0055 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0056     strcmp(struct_ver, '1.1')
0057
0058     % Set the struct_ver to '1.2'
0059     struct_ver = '1.1';
0060
0061 end
0062
0063 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0064
0065 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0066     strcmp(struct_ver, '1.2')
0067
0068     % Set the struct_ver to '1.3'
0069     struct_ver = '1.3';
0070
0071 end
0072
0073 varargout{1} = obj_struct;
0074
0075 end

```

[Home](#) > [classes](#) > [@ao](#) > welch.m

welch

PURPOSE ^

WELCH Welch spectral estimation method.

SYNOPSIS ^

function varargout = welch(varargin)

DESCRIPTION ^

WELCH Welch spectral estimation method.

```
[pxx, f, info] = welch(x,pl)
[pxx, f, info] = welch(x,y,pl)
```

PARAMETERS: 'Win' - a specwin window object [default: Kaiser -200dB psll]
 'Olap' - segment percent overlap [default: taken from window function]
 'Nfft' - number of samples in each fft [default: length of input data]
 'Scale' - one of

- 'ASD' - amplitude spectral density
- 'PSD' - power spectral density [default]
- 'AS' - amplitude spectrum
- 'PS' - power spectrum

* applies only to spectrum 'Type' 'psd'

'Order' - order of segment detrending

- 1 - no detrending
- 0 - subtract mean [default]
- 1 - subtract linear fit
- N - subtract fit of polynomial, order N

'Type' - type of spectrum:

- 'psd' - compute PSD
- 'cpsd' - compute cross-spectral density
- 'tfe' - estimate transfer function between inputs
- 'mscohere' - estimate cross-coherence

Copied directly from MATLAB and extended to do segment-wise detrending
 and to take a plist input

M Hewitson 08-05-08

\$Id: welch.m,v 1.5 2008/09/03 17:05:51 ingo Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [x](#) X Get the data property 'x'.
-  [xunits](#) XUNITS Get the data property 'xunits'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % WELCH Welch spectral estimation method.
0002 %
```

```

0003 % [pxx, f, info] = welch(x,pl)
0004 % [pxx, f, info] = welch(x,y,pl)
0005 %
0006 % PARAMETERS: 'Win' - a specwin window object [default: Kaiser -200dB psll]
0007 % 'Olap' - segment percent overlap [default: taken from window function]
0008 % 'Nfft' - number of samples in each fft [default: length of input data]
0009 % 'Scale' - one of
0010 %             'ASD' - amplitude spectral density
0011 %             'PSD' - power spectral density [default]
0012 %             'AS' - amplitude spectrum
0013 %             'PS' - power spectrum
0014 %             * applies only to spectrum 'Type' 'psd'
0015 % 'Order' - order of segment detrending
0016 %             -1 - no detrending
0017 %             0 - subtract mean [default]
0018 %             1 - subtract linear fit
0019 %             N - subtract fit of polynomial, order N
0020 % 'Type' - type of spectrum:
0021 %             'psd' - compute PSD
0022 %             'cpsd' - compute cross-spectral density
0023 %             'tfe' - estimate transfer function between inputs
0024 %             'mscohere' - estimate cross-coherence
0025 %
0026 % Copied directly from MATLAB and extended to do segment-wise detrending
0027 % and to take a plist input
0028 %
0029 % M Hewitson 08-05-08
0030 %
0031 % $Id: welch.m,v 1.5 2008/09/03 17:05:51 ingo Exp $
0032 %
0033 %
0034 % Author(s): P. Pacheco
0035 % Copyright 1988-2006 The MathWorks, Inc.
0036 % $Revision: 1.5 $ $Date: 2008/09/03 17:05:51 $
0037 %
0038 % References:
0039 % [1] Petre Stoica and Randolph Moses, Introduction To Spectral
0040 % Analysis, Prentice-Hall, 1997, pg. 15
0041 % [2] Monson Hayes, Statistical Digital Signal Processing and
0042 % Modeling, John Wiley & Sons, 1996.
0043 %
0044 % Compute the periodogram power spectrum of each segment and average always
0045 % compute the whole power spectrum, we force Fs = 1 to get a PS not a PSD.
0046 %
0047 function varargout = welch(varargin)
0048
0049 if nargin == 3
0050     a = varargin{1};
0051     esttype = varargin{2};
0052     pl = varargin{3};
0053     x = a.data.y;
0054     inunits = a.data.yunits;
0055 else
0056     a = varargin{1};
0057     b = varargin{2};
0058     esttype = varargin{3};
0059     pl = varargin{4};
0060     if a.data.fs ~= b.data.fs
0061         error('### Two time-series have different sample rates.');

```

```

0089 % Compute the periodogram power spectrum of each segment and average always
0090 % compute the whole power spectrum, we force Fs = 1 to get a PS not a PSD.
0091
0092 % Initialize
0093 if freqVectorSpecified,
0094     Sxx = zeros(length(options.nfft),1);
0095 else
0096     Sxx = zeros(options.nfft,1);
0097 end
0098 range = options.range;
0099
0100 LminusOverlap = L-noverlap;
0101 xStart = 1:LminusOverlap:k*LminusOverlap;
0102 xEnd = xStart+L-1;
0103 switch esttype,
0104 case {'ms','psd'},
0105     for i = 1:k,
0106         if order < 0
0107             seg = x(xStart(i):xEnd(i));
0108         else
0109             [seg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
0110         end
0111         [Sxxk,w] = ao.computeperiodogram(seg,win,...
0112             options.nfft,esttype,options.Fs);
0113         Sxx = Sxx + Sxxk;
0114     end
0115 case 'cpsd',
0116     for i = 1:k,
0117         if order < 0
0118             xseg = x(xStart(i):xEnd(i));
0119         else
0120             [xseg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
0121         end
0122         if order < 0
0123             yseg = y(xStart(i):xEnd(i));
0124         else
0125             [yseg,coeffs] = ltpda_polyreg(y(xStart(i):xEnd(i)), order);
0126         end
0127         [Sxxk,w] = ao.computeperiodogram({xseg,...
0128             yseg},win,options.nfft,esttype,options.Fs);
0129         Sxx = Sxx + Sxxk;
0130     end
0131 case 'tfe'
0132     Sxy = zeros(options.nfft,1); % Initialize
0133     for i = 1:k,
0134         if order < 0
0135             xseg = x(xStart(i):xEnd(i));
0136         else
0137             [xseg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
0138         end
0139         if order < 0
0140             yseg = y(xStart(i):xEnd(i));
0141         else
0142             [yseg,coeffs] = ltpda_polyreg(y(xStart(i):xEnd(i)), order);
0143         end
0144         [Sxxk,w] = ao.computeperiodogram(xseg,...
0145             win,options.nfft,esttype,options.Fs);
0146         [Syxk,w] = ao.computeperiodogram({yseg,...
0147             x(xStart(i):xEnd(i))},win,options.nfft,esttype,options.Fs);
0148         Sxx = Sxx + Sxxk;
0149         Sxy = Sxy + Syxk;
0150     end
0151 case 'mscohere'
0152     % Note: (Sxy1+Sxy2)/(Sxx1+Sxx2) != (Sxy1/Sxx1) + (Sxy2/Sxx2)
0153     % ie, we can't push the computation of Cxy into computeperiodogram.
0154     Sxy = zeros(options.nfft,1); % Initialize
0155     Syy = zeros(options.nfft,1); % Initialize
0156     for i = 1:k,
0157         if order < 0
0158             xseg = x(xStart(i):xEnd(i));
0159         else
0160             [xseg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
0161         end
0162         if order < 0
0163             yseg = y(xStart(i):xEnd(i));
0164         else
0165             [yseg,coeffs] = ltpda_polyreg(y(xStart(i):xEnd(i)), order);
0166         end
0167         [Sxxk,w] = ao.computeperiodogram(xseg,...
0168             win,options.nfft,esttype,options.Fs);
0169         [Syyk,w] = ao.computeperiodogram(yseg,...
0170             win,options.nfft,esttype,options.Fs);
0171         [Sxyk,w] = ao.computeperiodogram({xseg,...
0172             yseg},win,options.nfft,esttype,options.Fs);
0173         Sxx = Sxx + Sxxk;
0174         Syy = Syy + Syyk;

```



```

0175     Sxy = Sxy + Sxyk;
0176 end
0177 end
0178
0179 Sxx = Sxx./k; % Average the sum of the periodograms
0180 if any(strcmpi(esttype,{'tfe','mscohere'})),
0181     Sxy = Sxy./k; % Average the sum of the periodograms
0182
0183     if strcmpi(esttype,'mscohere'),
0184         Syy = Syy./k; % Average the sum of the periodograms
0185     end
0186 end
0187
0188 % Generate the freq vector directly in Hz to avoid roundoff errors due to
0189 % conversions later.
0190 if ~freqVectorSpecified,
0191     w = psdfreqvec('npts',options.nfft, 'Fs',options.Fs);
0192 else
0193     if strcmpi(options.range,'onesided')
0194         warning(generatemsgid('InconsistentRangeOption'),...
0195             'Ignoring ''onesided'' option. When a frequency vector is specified, a
0196             ''twosided'' PSD is computed');
0197     end
0198     options.range = 'twosided';
0199 end
0200
0201 % Compute the 1-sided or 2-sided PSD [Power/freq] or mean-square [Power].
0202 % Also, corresponding freq vector and freq units.
0203 [Pxx,w,units] = computepsd(Sxx,w,options.range,options.nfft,options.Fs,esttype);
0204
0205 if any(strcmpi(esttype,{'tfe','mscohere'})),
0206     % Cross PSD. The frequency vector and xunits are not used.
0207     [Pxy,f,xunits] = computepsd(Sxy,w,options.range,options.nfft,options.Fs,esttype);
0208
0209     % Transfer function estimate.
0210     if strcmpi(esttype,'tfe'),
0211         Pxx = Pxy ./ Pxx; % Txy
0212     end
0213
0214     % Magnitude Square Coherence estimate.
0215     if strcmpi(esttype,'mscohere'),
0216         % Auto PSD for 2nd input vector. The freq vector & xunits are not
0217         % used.
0218         [Pyy,f,xunits] = computepsd(Syy,w,options.range,options.nfft,options.Fs,esttype);
0219         Pxx = (abs(Pxy).^2)./(Pxx.*Pyy); % Cxy
0220     end
0221 end
0222
0223 % Scale to required units
0224 [Pxx, info] = ao.welchscale(Pxx, win, fs, scale, inunits);
0225 info.navs = k;
0226
0227 varargout = {Pxx, w, info};
0228 end
0229

```

welchparse

PURPOSE ^

WELCHPARSE Parser for the PWELCH & SPECTROGRAM functions

SYNOPSIS ^

function [x,M,isreal_x,y,Ly,win,winName,winParam,noverlap,k,L,options]
=welchparse(x,esttype,varargin)

DESCRIPTION ^

```
WELCHPARSE    Parser for the PWELCH & SPECTROGRAM functions

Outputs:
X             - First input signal (used when esttype = MS & PSD)
M             - An integer containing the length of the data to be segmented
isreal_x      - Boolean for input complexity
Y             - Second input signal (used when esttype = CPSD, TFE, MSCOHERE)
Ly            - Length of second input signal (used when esttype = CPSD, TFE, MSCOHERE)
WIN           - A scalar or vector containing the length of the window or the
                window respectively (Note that the length of the window determines
                the length of the segments)
WINNAME       - String with the window name.
WINPARAM      - Window parameter.
NOVERLAP      - An integer containing the number of samples to overlap (may
                be empty)
K             - Number of segments
OPTIONS       - A structure with the following fields:
OPTIONS.nfft   - number of freq. points at which the psd is estimated
OPTIONS.Fs     - sampling freq. if any
OPTIONS.range  - 'onesided' or 'twosided' psd

Direct copy from MATLAB

M Hewitson 08-05-08

$Id: welchparse.m,v 1.2 2008/08/01 13:19:43 ingo Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [max](#) MAX computes the maximum value of the data in the AO.
-  [x](#) X Get the data property 'x'.
-  [y](#) Y Get the data property 'y'.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SUBFUNCTIONS ^

-  [function \[x,Lx,isreal_x,y,Ly,win,winName,winParam,noverlap,opts,errid,errmsg\] =](#)
-  [function \[L,noverlap,win,errid,errmsg\] = segment_info\(M,win,noverlap\)](#)
-  [function \[options,msg\] = welch_options\(isreal_x,N,varargin\)](#)

SOURCE CODE ^

```

0001 % WELCHPARSE    Parser for the PWELCH & SPECTROGRAM functions
0002 %
0003 % Outputs:
0004 % X             - First input signal (used when esttype = MS & PSD)
0005 % M             - An integer containing the length of the data to be segmented
0006 % isreal_x      - Boolean for input complexity
0007 % Y             - Second input signal (used when esttype = CPSD, TFE, MSCOHERE)
0008 % Ly            - Length of second input signal (used when esttype = CPSD, TFE, MSCOHERE)
0009 % WIN            - A scalar or vector containing the length of the window or the
0010 %                 window respectively (Note that the length of the window determines
0011 %                 the length of the segments)
0012 % WINNAME        - String with the window name.
0013 % WINPARAM        - Window parameter.
0014 % NOVERLAP        - An integer containing the number of samples to overlap (may
0015 %                 be empty)
0016 % K              - Number of segments
0017 % OPTIONS        - A structure with the following fields:
0018 % OPTIONS.nfft     - number of freq. points at which the psd is estimated
0019 % OPTIONS.Fs        - sampling freq. if any
0020 % OPTIONS.range     - 'onesided' or 'twosided' psd
0021 %
0022 % Direct copy from MATLAB
0023 %
0024 % M Hewitson 08-05-08
0025 %
0026 % $Id: welchparse.m,v 1.2 2008/08/01 13:19:43 ingo Exp $
0027 %
0028
0029 %   Author(s): P. Costa
0030 %   Copyright 1988-2006 The MathWorks, Inc.
0031 %   $Revision: 1.2 $   $Date: 2008/08/01 13:19:43 $
0032
0033 function [x,M,isreal_x,y,Ly,win,winName,winParam,noverlap,k,L,options] = ...
0034     welchparse(x,esttype,varargin)
0035
0036 % Parse input arguments.
0037 [x,M,isreal_x,y,Ly,win,winName,winParam,noverlap,opts,errid,errmsg] = ...
0038     parse_inputs(x,esttype,varargin{:});
0039 if ~isempty(errmsg), error(errid,errmsg); end
0040
0041 % Obtain the necessary information to segment x and y.
0042 [L,noverlap,win,errid,errmsg] = segment_info(M,win,noverlap);
0043 if ~isempty(errmsg), error(errid,errmsg); end
0044
0045 % Parse optional args nfft, fs, and spectrumType.
0046 [options,msg] = welch_options(isreal_x,L,opts{:});
0047 if ~isempty(msg), error(generatemsgid('InvalidParam'),msg); end
0048
0049 % Compute the number of segments
0050 k = (M-noverlap)./(L-noverlap);
0051
0052 % Uncomment the following line to produce a warning each time the data
0053 % segmentation does not produce an integer number of segments.
0054 %if fix(k) ~= k,
0055 %    warning(generatemsgid('MustBeInteger'),'The number of segments is not an integer,
0056 %    truncating data. ');
0057 %end
0058 k = fix(k);
0059 end
0060
0061 %-----
0062
0062 function [x,Lx,isreal_x,y,Ly,win,winName,winParam,noverlap,opts,errid,errmsg] = ...
0063     parse_inputs(x,esttype,varargin)
0064 % Parse the inputs to the welch function.
0065
0066 % Assign defaults in case of early return.
0067 isreal_x = 1;
0068 y        = [];
0069 Ly       = 0;
0070 is2sig   = false;
0071 win      = [];
0072 winName  = 'User Defined';
0073 winParam = '';
0074 noverlap = [];
0075 opts     = {};
0076 errid    = '';
0077 errmsg   = '';
0078
0079 % Determine if one or two signal vectors was specified.
0080 Lx = length(x);
0081 if iscell(x),
0082     if Lx > 1, % Cell array.
0083         y = x{2};
0084         is2sig = true;

```

```

0085     end
0086     x = x{1};
0087     Lx = length(x);
0088     else
0089         if ~any(strcmpi(esttype,{'psd','ms'})),
0090             errid = generatemsgid('invalidSignalVectors');
0091             errmsg = 'You must specify a cell array with two signal vectors to estimate
either the cross power spectral density or the transfer function.';
0092             return;
0093         end
0094     end
0095
0096     x = x(:);
0097     isreal_x = isreal(x);
0098
0099     % Parse 2nd input signal vector.
0100     if is2sig,
0101         y = y(:);
0102         isreal_x = isreal(y) && isreal_x;
0103         Ly = length(y);
0104         if Ly ~= Lx,
0105             errid = generatemsgid('invalidSignalVectors');
0106             errmsg = 'The length of the two input vectors must be equal to calculate the
cross spectral density.';
0107             return;
0108         end
0109     end
0110
0111     % Parse window and overlap, and cache remaining inputs.
0112     lenargin = length(varargin);
0113     if lenargin >= 1,
0114         win = varargin{1};
0115         if lenargin >= 2,
0116             noverlap = varargin{2};
0117
0118             % Cache optional args nfft, fs, and spectrumType.
0119             if lenargin >= 3, opts = varargin(3:end); end
0120         end
0121     end
0122
0123     if isempty(win) | isscalar(win),
0124         winName = 'hamming';
0125         winParam = 'symmetric';
0126     end
0127
0128 end
0129
0130 %-----
0131 %SEGMENT_INFO    Determine the information necessary to segment the input data.
0132 %
0133 %    Inputs:
0134 %        M        - An integer containing the length of the data to be segmented
0135 %        WIN      - A scalar or vector containing the length of the window or the window
respectively
0136 %                  (Note that the length of the window determines the length of the
segments)
0137 %        NOVERLAP - An integer containing the number of samples to overlap (may be empty)
0138 %
0139 %    Outputs:
0140 %        L        - An integer containing the length of the segments
0141 %        NOVERLAP - An integer containing the number of samples to overlap
0142 %        WIN      - A vector containing the window to be applied to each section
0143 %        ERRID    - A string containing possible error identifier
0144 %        ERRMSG   - A string containing possible error messages
0145 %
0146 %
0147 %    The key to this function is the following equation:
0148 %
0149 %        
$$K = (M - NOVERLAP) / (L - NOVERLAP)$$

0150 %
0151 %    where
0152 %
0153 %        K        - Number of segments
0154 %        M        - Length of the input data X
0155 %        NOVERLAP - Desired overlap
0156 %        L        - Length of the segments
0157 %
0158 %    The segmentation of X is based on the fact that we always know M and two of the set
0159 %    {K, NOVERLAP, L}, hence determining the unknown quantity is trivial from the above
0160 %    formula.
0161
0162 function [L,noverlap,win,errid,errmsg] = segment_info(M,win,noverlap)
0163
0164 % Initialize outputs
0165 L = [];

```

```

0166 errid = '';
0167 errmsg = '';
0168
0169 % Check that noverlap is a scalar
0170 if any(size(noverlap) > 1),
0171     errid = generatemsgid('invalidNoverlap');
0172     errmsg = 'You must specify an integer number of samples to overlap.';
0173     return;
0174 end
0175
0176 if isempty(win),
0177     % Use 8 sections, determine their length
0178     if isempty(noverlap),
0179         % Use 50% overlap
0180         L = fix(M./4.5);
0181         noverlap = fix(0.5.*L);
0182     else
0183         L = fix((M+7.*noverlap)./8);
0184     end
0185     % Use a default window
0186     win = hamming(L);
0187 else
0188     % Determine the window and its length (equal to the length of the segments)
0189     if ~any(size(win) <= 1) | ischar(win),
0190         errid = generatemsgid('invalidWindow');
0191         errmsg = 'The WINDOW argument must be a vector or a scalar.';
0192         return;
0193     elseif length(win) > 1,
0194         % WIN is a vector
0195         L = length(win);
0196     elseif length(win) == 1,
0197         L = win;
0198         win = hamming(win);
0199     end
0200     if isempty(noverlap),
0201         % Use 50% overlap
0202         noverlap = fix(0.5.*L);
0203     end
0204 end
0205
0206 % Do some argument validation
0207 if L > M,
0208     errid = generatemsgid('invalidSegmentLength');
0209     errmsg = 'The length of the segments cannot be greater than the length of the input
signal.';
0210     return;
0211 end
0212
0213 if noverlap >= L,
0214     errid = generatemsgid('invalidNoverlap');
0215     errmsg = 'The number of samples to overlap must be less than the length of the
segments.';
0216     return;
0217 end
0218 end
0219
0220 %-----
0221 %WELCH_OPTIONS Parse the optional inputs to the PWELCH function.
0222 % WELCH_OPTIONS returns a structure, OPTIONS, with following fields:
0223 %
0224 % options.nfft - number of freq. points at which the psd is estimated
0225 % options.Fs - sampling freq. if any
0226 % options.range - 'onesided' or 'twosided' psd
0227
0228 function [options,msg] = welch_options(isreal_x,N,varargin)
0229
0230 % Generate defaults
0231 options.nfft = max(256,2^nextpow2(N));
0232 options.Fs = []; % Work in rad/sample
0233
0234 % Determine if frequency vector specified
0235 freqVecSpec = false;
0236 if (length(varargin) > 0 && length(varargin{1}) > 1)
0237     freqVecSpec = true;
0238 end
0239
0240 if isreal_x && ~freqVecSpec,
0241     options.range = 'onesided';
0242 else
0243     options.range = 'twosided';
0244 end
0245 msg = '';
0246
0247 if any(strcmp(varargin, 'whole'))
0248     warning(generatemsgid('invalidRange'), ''whole' is not a valid range, use
'twosided' instead.');
```

```
0249 elseif any(strcmp(varargin, 'half'))
0250     warning(generatemsgid('invalidRange'), ''half' is not a valid range, use
'onesided' instead. ');
0251 end
0252
0253 [options,msg] = psdoptions(isreal_x,options,varargin{:});
0254
0255 end
0256
```

[Home](#) > [classes](#) > [@ao](#) > welchscale.m

welchscale

PURPOSE [^]

WELCHSCALE scales the output of welch to be in the required units

SYNOPSIS [^]

```
function [yy, info] = welchscale(xx, win, fs, norm, inunits)
```

DESCRIPTION [^]

```
WELCHSCALE scales the output of welch to be in the required units
$Id: welchscale.m,v 1.5 2008/09/05 14:24:49 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [fs](#) FS Get the data property 'fs'.
-  [norm](#) NORM overloads the norm operator for Analysis Objects.
-  [sqrt](#) SQRT computes the square root of the data in the AO.
-  [sum](#) SUM computes the sum of the data in the AO.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE [^]

```

0001 % WELCHSCALE scales the output of welch to be in the required units
0002 %
0003 % $Id: welchscale.m,v 1.5 2008/09/05 14:24:49 hewitson Exp $
0004 %
0005
0006 function [yy, info] = welchscale(xx, win, fs, norm, inunits)
0007
0008     nfft = length(win);
0009     S1 = sum(win);
0010     S2 = sum(win.^2);
0011     enbw = fs * S2 / (S1*S1);
0012
0013     if isempty(norm)
0014         norm = 'None';
0015     end
0016     switch norm
0017     case 'ASD'
0018         yy = sqrt(xx);
0019         info.units = inunits ./ unit('Hz^0.5');
0020     case 'PSD'
0021         yy = xx;
0022         info.units = inunits.^2/unit('Hz');
0023     case 'AS'
0024         yy = sqrt(xx * enbw) ;
0025         info.units = inunits;
0026     case 'PS'
0027         yy = xx * enbw;
0028         info.units = inunits.^2;
0029     case 'None'
0030         yy = xx;
0031         info.units = inunits;
0032     otherwise
0033         error('Unknown normalisation');

```

```
0034     end
0035
0036     info.nfft = nfft;
0037     info.enbw = enbw;
0038     info.norm = norm;
0039
0040 end
0041
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003



PURPOSE ^

X Get the data property 'x'.

SYNOPSIS ^

function varargout = x(varargin)

DESCRIPTION ^

```
X Get the data property 'x'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the data property 'x'.

CALL:      val = ao.x();

INPUTS:    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('x')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('x', 'None')

VERSION:    $Id: x.m,v 1.1 2008/09/05 13:55:17 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- [ao](#) AO analysis object class constructor.
- [computeperiodogram](#) COMPUTEPERIODOGRAM Periodogram spectral estimation.
- [ctranspose](#) CTRANSPOSE overloads the ' operator for Analysis Objects.
- [detrend](#) DETREND detrends the input analysis object using a polynomial of degree N.
- [diff](#) DIFF differentiates the data in AO.
- [dsmean](#) DSMEAN performs a simple downsampling by taking the mean of every N samples.
- [find](#) FIND particular samples that satisfy the input query and return a new AO.
- [fngen](#) FNGEN creates an arbitrarily long time-series based on the input PSD.
- [fromDataInMAT](#) FROMDATAINMAT Convert a saved data-array into an AO with a tsdata-object
- [fromPzmodel](#) FROMPZMODEL Construct a time-series ao from polynomial coefficients
- [gapfilling](#) GAPFILLING fills possible gaps in data.
- [ge](#) GE overloads >= operator for analysis objects. Compare the y-axis values.
- [gt](#) GT overloads > operator for analysis objects. Compare the y-axis values.
- [hist](#) HIST overloads the histogram function (hist) of MATLAB for Analysis Objects.
- [ifft](#) IFFT overloads the ifft operator for Analysis objects.
- [iplot](#) IPLOT provides an intelligent plotting tool for LTPDA.
- [join](#) JOIN multiple AOs into a single AO.
- [le](#) LE overloads <= operator for analysis objects. Compare the y-axis values.

-  **lt** LT overloads < operator for analysis objects. Compare the y-axis values.
-  **mlpsd_m** MLPSD_M m-file only version of the LPSD algorithm
-  **mlpsd_mex** MLPSD_MEX calls the ltpda_dft.mex to compute the DFT part of the LPSD
-  **ngprop** NGPROP is called by the function LTPDA_NOISEGEN
-  **plot** PLOT a simple plot of analysis objects.
-  **polyfit** POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
-  **select** SELECT select particular samples from the input AOs and return new AOs with only those samples.
-  **setXY** SETXY sets the 'xy' property of the ao.
-  **split** SPLIT split an analysis object into the specified segments.
-  **timedomainfit** TIMEDOMAINFIT uses lscov to fit a set of time-series AOs to a target time-series AO.
-  **transpose** TRANSPOSE overloads the .' operator for Analysis Objects.
-  **welch** WELCH Welch spectral estimation method.
-  **welchparse** WELCHPARSE Parser for the PWELCH & SPECTROGRAM functions

SUBFUNCTIONS ^

-  **function ii = getInfo(varargin)**
-  **function plo = getDefaultPlist()**

SOURCE CODE ^

```

0001 % X Get the data property 'x'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the data property 'x'.
0005 %
0006 % CALL:      val = ao.x();
0007 %
0008 % INPUTS:    obj - must be a single data2D object.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %              >> ao.getInfo('x')
0012 %
0013 %              Get information about a specified set-plist by calling:
0014 %              >> ao.getInfo('x', 'None')
0015 %
0016 % VERSION:    $Id: x.m,v 1.1 2008/09/05 13:55:17 ingo Exp $
0017 %
0018 % HISTORY:    27-05-2008 Diepholz
0019 %              Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = x(varargin)
0024 %
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 obj = varargin{1};
0032 %
0033 if isprop(obj.data, 'x')
0034     varargout{1} = obj.data.getX;
0035 else
0036     varargout{1} = [];
0037 end
0038 %
0039 end
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %              Local Functions
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:    getInfo
0048 %
0049 % DESCRIPTION: Get Info Object

```

```
0050 %
0051 % HISTORY:      11-07-07 M Hewitson
0052 %              Creation.
0053 %
0054 %%%%%%%%%%%%%%
0055
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl    = [];
0060     else
0061         sets = {'Default'};
0062         pl    = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ao', '', utils.const.categories.internal, '$Id: x.m,v 1.1
2008/09/05 13:55:17 ingo Exp $', sets, pl);
0066 end
0067
0068 %%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:      getDefaultPlist
0071 %
0072 % DESCRIPTION:   Get Default Plist
0073 %
0074 % HISTORY:      11-07-07 M Hewitson
0075 %              Creation.
0076 %
0077 %%%%%%%%%%%%%%
0078
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082
```

xspec

PURPOSE ^

XSPEC performs cross-spectral analysis of various forms.

SYNOPSIS ^

function varargout = xspec(varargin)

DESCRIPTION ^

```
XSPEC performs cross-spectral analysis of various forms.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: XSPEC performs cross-spectral analysis of various forms.
             The function is a helper function for various higher level
             functions. It is meant to be called from other functions
             (e.g., ltpda_tfe).

CALL:       b = xspec(a, pl, method, iALGO, iVER, invars);

INPUTS:     a      - vector of input AOs
            pl      - input parameter list
            method  - one of
                     'cpsd'      - compute cross-spectral density
                     'tfe'      - estimate transfer function between inputs
                     'mscohere' - estimate cross-coherence
            iALGO   - ALGONAME from the calling higher level script
            iVER    - VERSION from the calling higher level script
            invars  - invars variable from the calling higher level script

OUTPUTS:    b      - Na x Na matrix of output AOs

VERSION:    $Id: xspec.m,v 1.15 2008/09/03 17:05:51 ingo Exp $

HISTORY:    30-05-2007 M Hewitson
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [abs](#) ABS overloads the Absolute value method for Analysis objects.
-  [ao](#) AO analysis object class constructor.
-  [copy](#) COPY makes a (deep) copy of the input AOs.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fs](#) FS Get the data property 'fs'.
-  [len](#) LEN overloads the length operator for Analysis objects. Length of the data samples.
-  [resample](#) RESAMPLE overloads resample function for AOs.

This function is called by:

-  [ao](#) AO analysis object class constructor.

SOURCE CODE ^

```
0001 % XSPEC performs cross-spectral analysis of various forms.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
```

```

0004 % DESCRIPTION: XSPEC performs cross-spectral analysis of various forms.
0005 %               The function is a helper function for various higher level
0006 %               functions. It is meant to be called from other functions
0007 %               (e.g., ltpda_tfe).
0008 %
0009 % CALL:         b = xspec(a, pl, method, iALGO, iVER, invars);
0010 %
0011 % INPUTS:       a      - vector of input AOs
0012 %               pl     - input parameter list
0013 %               method - one of
0014 %                   'cpsd' - compute cross-spectral density
0015 %                   'tfe'  - estimate transfer function between inputs
0016 %                   'mscohere' - estimate cross-coherence
0017 %               iALGO  - ALGONAME from the calling higher level script
0018 %               iVER   - VERSION from the calling higher level script
0019 %               invars - invars variable from the calling higher level script
0020 %
0021 % OUTPUTS:      b      - Na x Na matrix of output AOs
0022 %
0023 % VERSION:      $Id: xspec.m,v 1.15 2008/09/03 17:05:51 ingo Exp $
0024 %
0025 % HISTORY:      30-05-2007 M Hewitson
0026 %               Creation
0027 %
0028 %%%%%%%%%%%%%%%
0029
0030 function varargout = xspec(varargin)
0031
0032 import utils.const.*
0033
0034 % unpack inputs
0035 as = varargin{1};
0036 pl = varargin{2};
0037 method = varargin{3};
0038 mi = varargin{4};
0039 invars = varargin{5};
0040
0041 VERSION = '$Id: xspec.m,v 1.15 2008/09/03 17:05:51 ingo Exp $';
0042
0043 % Set the method version string in the minfo object
0044 mi.setMversion([VERSION '-->' mi.mversion]);
0045
0046 na = numel(as);
0047 if na < 2
0048     error('### XSPEC needs at least two time-series AOs.');
```

```

0090
0091 %----- check input parameters
0092
0093 % points in FFT
0094 Nfft = find(pl, 'Nfft');
0095 if ~isempty(Nfft)
0096     if Nfft < 0
0097         Nfft = abs(Nfft) * lmin;
0098         utils.helper.msg(msg.PROC2, 'using default Nfft of %g', Nfft);
0099         pl.pset('Nfft', Nfft);
0100     end
0101 end
0102
0103 % Window object
0104 Win = find(pl, 'Win');
0105 if ~isempty(Win)
0106     if length(Win.win)~=Nfft
0107         switch Win.type
0108             case 'Kaiser'
0109                 Win = specwin(Win.type, Nfft, Win.psl1);
0110             otherwise
0111                 Win = specwin(Win.type, Nfft);
0112         end
0113         utils.helper.msg(msg.PROC2, 'reset window to %s(%d)', strrep(Win.type, '_', '\_'),
length(Win.win));
0114         pl.pset('Win', Win);
0115     end
0116 end
0117
0118 % desired overlap
0119 Olap = find(pl, 'Olap');
0120 if ~isempty(Olap)
0121     if Olap < 0
0122         % use window function rov
0123         Olap = Win.rov;
0124         utils.helper.msg(msg.PROC2, 'using recommended overlap of %2.1f%%', Olap);
0125         pl.pset('Olap', Olap);
0126     end
0127 end
0128
0129 % desired detrending order
0130 order = find(pl, 'Order');
0131 if isempty(order)
0132     order = 0;
0133     utils.helper.msg(msg.PROC1, 'using default detrending order 0 (mean)');
0134     pl.append('Order', order);
0135 end
0136
0137 % Loop over input AOs
0138 bs(na,na) = ao;
0139 for j=1:na
0140     for k=1:na
0141         if k>=j
0142             utils.helper.msg(msg.PROC1, 'computing (%d,%d) %s -> %s', j, k, invars{j},
invars{k});
0143
0144             % ----- Make Xspec estimate
0145
0146             % Compute xspec using welch
0147             [txy, f, info] = ao.welch(as(j), as(k), method, pl);
0148
0149             % Keep the data shape of the first input A0
0150             if size(as(1).data.y,1) == 1
0151                 txy = txy.';
0152                 f = f.';
0153             end
0154
0155             % create new output fsdata
0156             fs = fsdata(f, txy, fsmax);
0157             fs.setEnbw(info.enbw);
0158             fs.setXunits('Hz');
0159             fs.setYunits(info.units);
0160             fs.setNavs(info.navs);
0161
0162             % make output analysis object
0163             bs(j,k) = ao(fs);
0164
0165             %----- Add history
0166
0167             % we need to get the input histories in the same order as the inputs
0168             % to this function call, not in the order of the input to tfestimate;
0169             % otherwise the resulting matrix on a 'create from history' will be
0170             % mirrored.
0171             if j<k
0172                 bs(j,k).addHistory(mi, pl, [invars(j) invars(k)], [as(j).hist, as(k).hist]);
0173             else

```

```
0174         bs(j,k).addHistory(mi, pl, [invars(j) invars(k)], [as(k).hist, as(j).hist]);
0175     end
0176
0177     % set name
0178     bs(j,k).setName(sprintf( '%s(%s->%s)', method, invars{j}, invars{k}),
'internal');
0179
0180     end % End upper triangle
0181 end % End second loop over A0s
0182 end % End first loop over A0s
0183
0184 % Fill remainder
0185 for j=1:na
0186     for k=1:na
0187         if k<j
0188             % Do we need a copy here? Perhaps
0189             % copying the pointer is enough...
0190             bs(j,k) = bs(k,j);
0191         end
0192     end
0193 end
0194
0195 % Set output
0196 varargout{1} = bs;
0197 end
0198
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

xunits

PURPOSE ^

XUNITS Get the data property 'xunits'.

SYNOPSIS ^

function varargout = xunits(varargin)

DESCRIPTION ^

```
XUNITS Get the data property 'xunits'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the data property 'xunits'.

CALL:      val = ao.xunits();

INPUTS:    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('xunits')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('xunits', 'None')

VERSION:    $Id: xunits.m,v 1.1 2008/09/05 13:55:17 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  **ao** AO analysis object class constructor.
-  **fft** FFT overloads the fft method for Analysis objects.
-  **join** JOIN multiple AOs into a single AO.
-  **welch** WELCH Welch spectral estimation method.

SUBFUNCTIONS ^

-  **function** ii = getInfo(varargin)
-  **function** plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % XUNITS Get the data property 'xunits'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the data property 'xunits'.
0005 %
0006 % CALL:      val = ao.xunits();
0007 %
0008 % INPUTS:    obj - must be a single data2D object.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
```



```
0011 %           >> ao.getInfo('xunits')
0012 %
0013 %           Get information about a specified set-plist by calling:
0014 %           >> ao.getInfo('xunits', 'None')
0015 %
0016 % VERSION:      $Id: xunits.m,v 1.1 2008/09/05 13:55:17 ingo Exp $
0017 %
0018 % HISTORY:      27-05-2008 Diepholz
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = xunits(varargin)
0024 %
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030 %
0031 obj = varargin{1};
0032 %
0033 if isprop(obj.data, 'xunits')
0034     varargout{1} = obj.data.xunits;
0035 else
0036     varargout{1} = [];
0037 end
0038 %
0039 end
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %                                Local Functions                                %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044 %
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:      getInfo
0048 %
0049 % DESCRIPTION:   Get Info Object
0050 %
0051 % HISTORY:      11-07-07 M Hewitson
0052 %                Creation.
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl    = [];
0060     else
0061         sets = {'Default'};
0062         pl    = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'ao', '', utils.const.categories.internal, '$Id: xunits.m,v 1.1
2008/09/05 13:55:17 ingo Exp $', sets, pl);
0066 end
0067 %
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:      getDefaultPlist
0071 %
0072 % DESCRIPTION:   Get Default Plist
0073 %
0074 % HISTORY:      11-07-07 M Hewitson
0075 %                Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078 %
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082 %
```

y

PURPOSE ^

Y Get the data property 'y'.

SYNOPSIS ^

function varargout = y(varargin)

DESCRIPTION ^

```
Y Get the data property 'y'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the data property 'y'.

CALL:      val = ao.y();

INPUTS:    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('y')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('y', 'None')

VERSION:   $Id: y.m,v 1.1 2008/09/05 13:55:17 ingo Exp $

HISTORY:   27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [computeperiodogram](#) COMPUTEPERIODOGRAM Periodogram spectral estimation.
-  [ctranspose](#) CTRANSPOSE overloads the ' operator for Analysis Objects.
-  [detrend](#) DETREND detrends the input analysis object using a polynomial of degree N.
-  [diff](#) DIFF differentiates the data in AO.
-  [dsmean](#) DSMEAN performs a simple downsampling by taking the mean of every N samples.
-  [find](#) FIND particular samples that satisfy the input query and return a new AO.
-  [fromFSfcn](#) FROMFSFCN Construct an ao from a fs-function string
-  [fromPolyval](#) FROMPOLYVAL Construct a time-series ao from polynomial coefficients
-  [fromPzmodel](#) FROMPZMODEL Construct a time-series ao from polynomial coefficients
-  [fromTSfcn](#) FROMTSFCN Construct an ao from a ts-function string
-  [fromWaveform](#) FROMWAVEFORM Construct an ao from a waveform
-  [ge](#) GE overloads >= operator for analysis objects. Compare the y-axis values.
-  [gt](#) GT overloads > operator for analysis objects. Compare the y-axis values.
-  [ifft](#) IFFT overloads the ifft operator for Analysis objects.
-  [iplot](#) IPLOT provides an intelligent plotting tool for LTPDA.
-  [le](#) LE overloads <= operator for analysis objects. Compare the y-axis values.
-  [lincom](#) LINCOM

-  [linedetect](#) LINEDETECT find spectral lines in the ao/fsdata objects.
-  [lt](#) LT overloads < operator for analysis objects. Compare the y-axis values.
-  [lxspe](#) LXSPEC performs log-scale cross-spectral analysis of various forms.
-  [nginit](#) NGINIT is called by the function LTPDA_NOISEGEN
-  [ngprop](#) NGPROP is called by the function LTPDA_NOISEGEN
-  [plot](#) PLOT a simple plot of analysis objects.
-  [polyfit](#) POLYFIT overloads polyfit() function of MATLAB for Analysis Objects.
-  [setXY](#) SETXY sets the 'xy' property of the ao.
-  [smoother](#) SMOOTHER smooths a given series of data points using the specified method.
-  [split](#) SPLIT split an analysis object into the specified segments.
-  [transpose](#) TRANSPOSE overloads the '.' operator for Analysis Objects.
-  [welch](#) WELCH Welch spectral estimation method.
-  [welchparse](#) WELCHPARSE Parser for the PWELCH & SPECTROGRAM functions

SUBFUNCTIONS ^

-  `function ii = getInfo(varargin)`
-  `function plo = getDefaultPlist()`

SOURCE CODE ^

```

0001 % Y Get the data property 'y'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the data property 'y'.
0005 %
0006 % CALL:      val = ao.y();
0007 %
0008 % INPUTS:    obj - must be a single data2D object.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %              >> ao.getInfo('y')
0012 %
0013 %              Get information about a specified set-plist by calling:
0014 %              >> ao.getInfo('y', 'None')
0015 %
0016 % VERSION:   $Id: y.m,v 1.1 2008/09/05 13:55:17 ingo Exp $
0017 %
0018 % HISTORY:   27-05-2008 Diepholz
0019 %              Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = y(varargin)
0024
0025 %%% Check if this is a call for parameters
0026 if utils.helper.isinfocall(varargin{:})
0027     varargout{1} = getInfo(varargin{3});
0028     return
0029 end
0030
0031 obj = varargin{1};
0032
0033 varargout{1} = obj.data.getY;
0034
0035 end
0036
0037 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0038 %                                Local Functions                                %
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %
0043 % FUNCTION:   getInfo
0044 %
0045 % DESCRIPTION: Get Info Object
0046 %
0047 % HISTORY:   11-07-07 M Hewitson
0048 %              Creation.
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %

```

```
0052 function ii = getInfo(varargin)
0053     if nargin == 1 && strcmpi(varargin{1}, 'None')
0054         sets = {};
0055         pl = [];
0056     else
0057         sets = {'Default'};
0058         pl = getDefaultPlist;
0059     end
0060     % Build info object
0061     ii = minfo(mfilename, 'ao', '', utils.const.categories.internal, '$Id: y.m,v 1.1
2008/09/05 13:55:17 ingo Exp $', sets, pl);
0062 end
0063
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065 %
0066 % FUNCTION:      getDefaultPlist
0067 %
0068 % DESCRIPTION:  Get Default Plist
0069 %
0070 % HISTORY:      11-07-07 M Hewitson
0071 %               Creation.
0072 %
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074
0075 function plo = getDefaultPlist()
0076     plo = plist();
0077 end
0078
```

yunits

PURPOSE ^

YUNITS Get the data property 'yunits'.

SYNOPSIS ^

function varargout = yunits(varargin)

DESCRIPTION ^

```
YUNITS Get the data property 'yunits'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Get the data property 'yunits'.

CALL:      val = ao.yunits();

INPUTS:    obj - must be a single data2D object.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('yunits')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('yunits', 'None')

VERSION:    $Id: yunits.m,v 1.1 2008/09/05 13:55:17 ingo Exp $

HISTORY:    27-05-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ao](#) AO analysis object class constructor.
-  [dft](#) DFT computes the DFT of the input time-series at the requested frequencies.
-  [join](#) JOIN multiple AOs into a single AO.

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % YUNITS Get the data property 'yunits'.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Get the data property 'yunits'.
0005 %
0006 % CALL:      val = ao.yunits();
0007 %
0008 % INPUTS:    obj - must be a single data2D object.
0009 %
0010 % M-FILE INFO: Get information about this methods by calling
0011 %              >> ao.getInfo('yunits')
```

```
0012 %
0013 %           Get information about a specified set-plist by calling:
0014 %           >> ao.getInfo('yunits', 'None')
0015 %
0016 % VERSION:      $Id: yunits.m,v 1.1 2008/09/05 13:55:17 ingo Exp $
0017 %
0018 % HISTORY:      27-05-2008 Diepholz
0019 %                Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022
0023 function varargout = yunits(varargin)
0024
0025     %% Check if this is a call for parameters
0026     if utils.helper.isinfocall(varargin{:})
0027         varargout{1} = getInfo(varargin{3});
0028         return
0029     end
0030
0031     obj = varargin{1};
0032
0033     varargout{1} = obj.data.yunits;
0034
0035 end
0036
0037 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0038 %                                Local Functions                                %
0039 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %
0043 % FUNCTION:      getInfo
0044 %
0045 % DESCRIPTION:   Get Info Object
0046 %
0047 % HISTORY:      11-07-07 M Hewitson
0048 %                Creation.
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051
0052 function ii = getInfo(varargin)
0053     if nargin == 1 && strcmpi(varargin{1}, 'None')
0054         sets = {};
0055         pl = [];
0056     else
0057         sets = {'Default'};
0058         pl = getDefaultPlist;
0059     end
0060     % Build info object
0061     ii = minfo(mfilename, 'ao', '', utils.const.categories.internal, '$Id: yunits.m,v 1.1
2008/09/05 13:55:17 ingo Exp $', sets, pl);
0062 end
0063
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065 %
0066 % FUNCTION:      getDefaultPlist
0067 %
0068 % DESCRIPTION:   Get Default Plist
0069 %
0070 % HISTORY:      11-07-07 M Hewitson
0071 %                Creation.
0072 %
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074
0075 function plo = getDefaultPlist()
0076     plo = plist();
0077 end
0078
```

fromFile

PURPOSE ^

Construct a ltpda_ob from a file

SYNOPSIS ^

function objs = fromFile(obj, pli)

DESCRIPTION ^

```
Construct a ltpda_ob from a file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromFile

DESCRIPTION: Construct a ltpda_ob from a file

CALL:         obj = obj.fromFile(filename)
              obj = obj.fromFile(pl)

VERSION:      $Id: fromFile.m,v 1.2 2008/09/08 09:40:46 hewitson Exp $

HISTORY:      21-08-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [find](#) FIND overloads find routine for a parameter list.
-  [plist](#) PLIST Plist class object constructor.

This function is called by:

-  [plist](#) PLIST Plist class object constructor.

SOURCE CODE ^

```
0001 % Construct a ltpda_ob from a file
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromFile
0005 %
0006 % DESCRIPTION: Construct a ltpda_ob from a file
0007 %
0008 % CALL:         obj = obj.fromFile(filename)
0009 %              obj = obj.fromFile(pl)
0010 %
0011 % VERSION:      $Id: fromFile.m,v 1.2 2008/09/08 09:40:46 hewitson Exp $
0012 %
0013 % HISTORY:      21-08-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function objs = fromFile(obj, pli)
0018
0019     VERSION = '$Id: fromFile.m,v 1.2 2008/09/08 09:40:46 hewitson Exp $';
0020
0021     % Which file type are we dealing with?
0022     if ischar(pli)
0023         pli = plist('filename', pli);
```

```

0024 end
0025
0026 % get filename
0027 filename = find(pli, 'filename');
0028
0029 % Get the correct parameter set
0030 [path, name, ext] = fileparts(filename);
0031
0032 switch ext
0033     case '.mat'
0034         ii = obj.getInfo(class(obj), 'From MAT File');
0035         ii.setMversion([VERSION '-->' ii.mversion]);
0036         objs = load(filename);
0037
0038         if isstruct(objs) && isfield(objs, 'objs') % check the class and the
struct match
0039             scl = utils.helper.classFromStruct(objs.objs);
0040             if isempty(scl)
0041                 error('### The structure does not match any LTPDA object.');
```

0042 end

0043 if ~strcmp('plist', scl)

0044 error('### The structure does not match the chosen LTPDA object constructor.
It seems to be a [%s] object.', scl)

0045 end

0046 objs = plist(objs.objs);

0047 elseif ismember('a', fieldnames(objs))

0048 objs = objs.a;

0049 elseif ismember('objs', fieldnames(objs))

0050 objs = objs.objs;

0051 else

0052 error('### MAT file contents can not be interpreted.');

0053 end

0054 case '.xml'

0055 root_node = xmlread(filename);

0056 objs = utils.helper.xmlread(root_node, class(obj));

0057 otherwise

0058 error('### Unknown file type [%s].', ext(2:end));

0059 end

0060 end

0061 end

0062 end

0063 end

0064

fromRepository

PURPOSE ^

Construct a plist filter from a repository

SYNOPSIS ^

function fs = fromRepository(f, pli)

DESCRIPTION ^

```
Construct a plist filter from a repository
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromRepository
DESCRIPTION: Construct a plist filter from a repository
CALL:         f = fromRepository(pl)
PARAMETER:    pl:      Parameter list object
HISTORY:      07-05-2007 Hewitson
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [combine](#) COMBINE multiple parameter lists (plist objects) into a single plist.
-  [find](#) FIND overloads find routine for a parameter list.

This function is called by:

-  [plist](#) PLIST Plist class object constructor.

SOURCE CODE ^

```
0001 % Construct a plist filter from a repository
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromRepository
0005 %
0006 % DESCRIPTION: Construct a plist filter from a repository
0007 %
0008 % CALL:         f = fromRepository(pl)
0009 %
0010 % PARAMETER:    pl:      Parameter list object
0011 %
0012 %
0013 % HISTORY:      07-05-2007 Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function fs = fromRepository(f, pli)
0018
0019     VERSION = '$Id: fromRepository.m,v 1.2 2008/09/04 15:29:31 ingo Exp $';
0020     % get A0 info
0021     ii = mfir.getInfo('plist', 'From Repository');
0022     % Set the method version string in the minfo object
0023     ii.setMversion([VERSION '-->' ii.mversion]);
```

```

0024
0025 % Add default values
0026 pl = combine(pli, ii.plist);
0027
0028 % Get parameters
0029 conn = find(pl, 'conn');
0030 hostname = find(pl, 'hostname');
0031 database = find(pl, 'database');
0032 ids = find(pl, 'id');
0033
0034 % do we have a connection?
0035 closeConn = 0;
0036 if isempty(conn)
0037     closeConn = 1;
0038     % Connect to repository
0039     conn = utils.mysql.connect(hostname, database);
0040 end
0041 if ~isa(conn, 'database')
0042     error('### connection failed.');
```

```

0043 end
0044
0045 % Get each ID
0046 Nids = length(ids);
0047 fs = [];
0048 for kk=1:Nids
0049
0050     %---- This id
0051     id = ids(kk);
0052     utils.helper.msg(utils.const.msg.OPROC2, 'retrieving ID %d', id);
0053
0054     %---- check ID object type
0055     tt = utils.mysql.getObjType(conn, id);
0056
0057     %---- If this is an AO
0058     if strcmp(tt, 'plist')
0059         %---- call database constructor
0060         f = ltpda_uo.retrieve(conn, id);
0061         %---- Add history
0062
0063         % make a new history object
0064         % - remove connection object from plist first
0065         f.addHistory(ii, pl, [], []);
0066
0067         %---- Add to output array
0068         fs = [fs f];
0069     else
0070         warning('      !skipping ID %d, type %s', id, tt);
0071     end
0072 end
0073
0074
0075 % close connection
0076 if closeConn
0077     close(conn);
0078 end
0079 end

```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [plist](#) PLIST Plist class object constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035         obj_struct = rmfield(obj_struct, 'plist');
0036         obj_struct.creator = provenance('updater 1.0');
0037     catch ME
0038         disp(varargin{1});
0039         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (plist) above is not
from version 1.0')));
0040     end
0041
0042     % Set the struct_ver to '1.1'
0043     struct_ver = '1.1';
0044
0045 end
0046
0047 %%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%
0048
0049 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0050     strcmp(struct_ver, '1.1')
0051
0052     % Set the struct_ver to '1.2'
0053     struct_ver = '1.1';
0054
0055 end
0056
0057 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0058
0059 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0060     strcmp(struct_ver, '1.2')
0061
0062     % Set the struct_ver to '1.3'
0063     struct_ver = '1.3';
0064
0065 end
0066
0067 varargout{1} = obj_struct;
0068

```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [param](#) PARAM Parameter object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035         obj_struct = rmfield(obj_struct, 'name');
0036         obj_struct = rmfield(obj_struct, 'created');
0037     catch ME
0038         disp(varargin{1});
0039         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (param) above is not
from version 1.0')));
0040     end
0041
0042     % Set the struct_ver to '1.1'
0043     struct_ver = '1.1';
0044
0045 end
0046
0047 %%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%
0048
0049 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0050     strcmp(struct_ver, '1.1')
0051
0052     % Set the struct_ver to '1.2'
0053     struct_ver = '1.1';
0054
0055 end
0056
0057 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0058
0059 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0060     strcmp(struct_ver, '1.2')
0061
0062     % Set the struct_ver to '1.3'
0063     struct_ver = '1.3';
0064
0065 end
0066
0067 varargout{1} = obj_struct;
0068

```

abccascade

PURPOSE ^

ABCCASCADE Cascade two filters together to get a new filter.

SYNOPSIS ^

function [ao,bo] = abccascade(a1,b1,a2,b2)

DESCRIPTION ^

ABCCASCADE Cascade two filters together to get a new filter.
DESCRIPTION: ABCCASCADE Cascade two filters together to get a new filter.
CALL: [a,b] = abccascade(a1,b1,a2,b2)
VERSION: \$Id: abccascade.m,v 1.2 2008/07/22 10:24:00 ingo Exp \$
HISTORY: 03-04-2007 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SOURCE CODE ^

```
0001 % ABCCASCADE Cascade two filters together to get a new filter.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: ABCCASCADE Cascade two filters together to get a new filter.
0005 %
0006 % CALL: [a,b] = abccascade(a1,b1,a2,b2)
0007 %
0008 % VERSION: $Id: abccascade.m,v 1.2 2008/07/22 10:24:00 ingo Exp $
0009 %
0010 % HISTORY: 03-04-2007 M Hewitson
0011 % Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014 %
0015 function [ao,bo] = abccascade(a1,b1,a2,b2)
0016
0017     n = length(a1);
0018     m = length(a2);
0019     N = m+n-1;
0020
0021     alpha = zeros(1, N);
0022     beta = zeros(1, N);
0023
0024     for i=0:n-1
0025         for j=0:m-1
0026             alpha(i+j+1) = alpha(i+j+1) + a1(i+1)*a2(j+1);
0027             beta(i+j+1) = beta(i+j+1) + b1(i+1)*b2(j+1);
0028         end
0029     end
0030
0031     ao = alpha;
```

```
0032    bo = beta;  
0033  end  
0034
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromLISO

PURPOSE ^

FROMLISO Construct a pzmodel from a LISO file

SYNOPSIS ^

function pzm = fromLISO(pzm, pli)

DESCRIPTION ^

```
FROMLISO Construct a pzmodel from a LISO file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromLISO
DESCRIPTION: Construct a pzmodel from a LISO file
CALL:         pzm = fromLISO(pzm, pli)
PARAMETER:    pzm: Empty pole/zero model
              pli: input plist (must contain the filename)
VERSION:      $Id: fromLISO.m,v 1.3 2008/08/22 14:17:27 ingo Exp $
HISTORY:      22-03-2008 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SUBFUNCTIONS ^

 [function d = lisoStr2Num\(s\)](#)

SOURCE CODE ^

```
0001 % FROMLISO Construct a pzmodel from a LISO file
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromLISO
0005 %
0006 % DESCRIPTION: Construct a pzmodel from a LISO file
0007 %
0008 % CALL:         pzm = fromLISO(pzm, pli)
0009 %
0010 % PARAMETER:    pzm: Empty pole/zero model
0011 %              pli: input plist (must contain the filename)
0012 %
0013 % VERSION:      $Id: fromLISO.m,v 1.3 2008/08/22 14:17:27 ingo Exp $
0014 %
0015 % HISTORY:      22-03-2008 M Hewitson
0016 %              Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
```

```

0020 function pzm = fromLISO(pzm, pli)
0021
0022     import utils.const.*
0023
0024     VERSION = '$Id: fromLISO.m,v 1.3 2008/08/22 14:17:27 ingo Exp $';
0025
0026     ii = pzm.getInfo(class(pzm), 'From LISO File');
0027
0028     % Set the method version string in the minfo object
0029     ii.setMversion([VERSION '-->' ii.mversion]);
0030
0031     filename = find(pli, 'filename');
0032
0033     poles = [];
0034     zeros = [];
0035     gain = [];
0036     % Open the file for reading
0037     fd = fopen(filename, 'r');
0038     utils.helper.msg(msg.OPROC2, ['reading ' filename]);
0039     while ~feof(fd)
0040         line = fgetl(fd);
0041         % get first token
0042         [s,r] = strtok(line);
0043         if strcmp(s, 'pole')
0044             % get next token as frequency
0045             [s,r] = strtok(r);
0046             pf = lisoStr2Num(s);
0047             % does this pole have a Q?
0048             pq = NaN;
0049             if ~isempty(r)
0050                 [s,r] = strtok(r);
0051                 if isnumeric(lisoStr2Num(s))
0052                     pq = lisoStr2Num(s);
0053                 end
0054             end
0055             % make pole
0056             if isnan(pq)
0057                 p = pz(pf);
0058                 utils.helper.msg(msg.OPROC2, 'found pole: %g', pf);
0059             else
0060                 p = pz(pf, pq);
0061                 utils.helper.msg(msg.OPROC2, 'found pole: %g, %g', pf, pq);
0062             end
0063             poles = [poles p];
0064         elseif strcmp(s, 'zero')
0065             % get next token as frequency
0066             [s,r] = strtok(r);
0067             zf = lisoStr2Num(s);
0068             % does this pole have a Q?
0069             zq = NaN;
0070             if ~isempty(r)
0071                 [s,r] = strtok(r);
0072                 if isnumeric(lisoStr2Num(s))
0073                     zq = lisoStr2Num(s);
0074                 end
0075             end
0076             % make zero
0077             if isnan(zq)
0078                 z = pz(zf);
0079                 utils.helper.msg(msg.OPROC2, 'found zero: %g', zf);
0080             else
0081                 z = pz(zf, zq);
0082                 utils.helper.msg(msg.OPROC2, 'found zero: %g, %g', zf, zq);
0083             end
0084             zeros = [zeros z];
0085         elseif strcmp(s, 'factor')
0086             % get next token as gain
0087             [s, r] = strtok(r);
0088             gain = lisoStr2Num(s);
0089             utils.helper.msg(msg.OPROC2, 'found factor: %g', gain);
0090         end
0091     end
0092     % Close file
0093     fclose(fd);
0094     % get model name
0095     [path, name, ext, vers] = fileparts(filename);
0096     % set object
0097     pzm.gain = gain;
0098     pzm.poles = poles;
0099     pzm.zeros = zeros;
0100     pzm.name = name;
0101
0102     pzm.addHistory(ii, pli, [], []);
0103
0104 end
0105

```

```
0106 %%%%%%%%% A function to convert LISO number strings to doubles %%%%%%%%%
0107
0108 function d = lisoStr2Num(s)
0109
0110     s = strrep(s, 'm', 'e-3');
0111     s = strrep(s, 'u', 'e-6');
0112     s = strrep(s, 'n', 'e-9');
0113     s = strrep(s, 'p', 'e-12');
0114     s = strrep(s, 'f', 'e-15');
0115     s = strrep(s, 'k', 'e3');
0116     s = strrep(s, 'M', 'e6');
0117     s = strrep(s, 'G', 'e9');
0118
0119     d = str2double(s);
0120
0121 end % function d = lisoStr2Num(s)
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromPolesAndZeros

PURPOSE ^

%%%

SYNOPSIS ^

```
function pzm = fromPolesAndZeros(pzm, pli)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromPolesAndZeros
DESCRIPTION:   Construct a pzmodel from poles and zeros
CALL:         pzm = fromPolesAndZeros(a, pl)
PARAMETER:    pl    - plist
HISTORY:      07-05-2007 Hewitson
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      fromPolesAndZeros
0004 %
0005 % DESCRIPTION:   Construct a pzmodel from poles and zeros
0006 %
0007 % CALL:         pzm = fromPolesAndZeros(a, pl)
0008 %
0009 % PARAMETER:
```

```

0010 %           pl    - plist
0011 %
0012 % HISTORY:    07-05-2007 Hewitson
0013 %           Creation
0014 %
0015 %%%%%%%%%%%
0016 function pzm = fromPolesAndZeros(pzm, pli)
0017
0018     VERSION = '$Id: fromPolesAndZeros.m,v 1.9 2008/09/04 13:37:14 ingo Exp $';
0019
0020
0021 % get pzmodel info
0022 ii = pzmodel.getInfo('pzmodel', 'From Poles/Zeros');
0023
0024 % Set the method version string in the minfo object
0025 ii.setMversion([VERSION '-->' ii.mversion]);
0026
0027 % Combine input plist with default values
0028 pl = combine(pli, ii.plist);
0029
0030 % Set fields
0031 pzm.gain = find(pl, 'gain');
0032 ps = find(pl, 'poles');
0033 zs = find(pl, 'zeros');
0034
0035 % If the poles/zeros in the plist are no pz-objects then convert them
0036 if ~isa(ps, 'pz') && ~isempty(ps)
0037     ps = pz(ps);
0038 end
0039 if ~isa(zs, 'pz') && ~isempty(zs)
0040     zs = pz(zs);
0041 end
0042
0043 % Add only valid poles
0044 pzm.poles = [];
0045 for kk = 1:length(ps)
0046     if ~isnan(ps(kk).f)
0047         pzm.poles = [pzm.poles ps(kk)];
0048     end
0049 end
0050
0051 % Add only valid zeros
0052 pzm.zeros = [];
0053 for kk = 1:length(zs)
0054     if ~isnan(zs(kk).f)
0055         pzm.zeros = [pzm.zeros zs(kk)];
0056     end
0057 end
0058
0059 name = find(pl, 'name');
0060 if ~isempty(name)
0061     pzm.name = name;
0062 end
0063
0064 % Add history
0065 pzm.addHistory(ii, pl, [], []);
0066
0067 end

```

setGain

PURPOSE ^

SETGAIN Set the property 'gain'

SYNOPSIS ^

function obj = setGain(obj, val)

DESCRIPTION ^

```
SETGAIN Set the property 'gain'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETGAIN Set the property 'gain'

CALL:      obj = obj.setGain(pz(2));
           obj = setGain(obj, pz(2));

INPUTS:    obj - is a miir object

VERSION:    $Id: setGain.m,v 1.1 2008/07/22 10:24:01 ingo Exp $

HISTORY:    16-07-2008 Diepholz
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

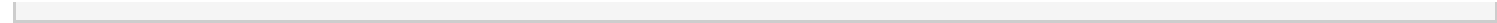
 [copy](#) COPY Make copy of pzmodel objects depending of the second input

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SOURCE CODE ^

```
0001 % SETGAIN Set the property 'gain'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETGAIN Set the property 'gain'
0005 %
0006 % CALL:      obj = obj.setGain(pz(2));
0007 %           obj = setGain(obj, pz(2));
0008 %
0009 % INPUTS:    obj - is a miir object
0010 %
0011 % VERSION:    $Id: setGain.m,v 1.1 2008/07/22 10:24:01 ingo Exp $
0012 %
0013 % HISTORY:    16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function obj = setGain(obj, val)
0019     %% decide whether we modify the pzmodel-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'gain'
0023     obj.gain = val;
0024 end
0025
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setPoles

PURPOSE ^

SETPOLES Set the property 'poles'

SYNOPSIS ^

function obj = setPoles(obj, val)

DESCRIPTION ^

```
SETPOLES Set the property 'poles'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETPOLES Set the property 'poles'

CALL:      obj = obj.setPoles(pz(2));
           obj = setPoles(obj, pz(2));

INPUTS:    obj - is a miir object

VERSION:    $Id: setPoles.m,v 1.1 2008/07/22 10:24:01 ingo Exp $

HISTORY:    16-07-2008 Diepholz
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

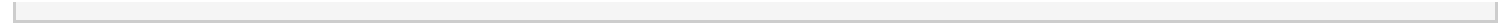
 [copy](#) COPY Make copy of pzmodel objects depending of the second input

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SOURCE CODE ^

```
0001 % SETPOLES Set the property 'poles'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETPOLES Set the property 'poles'
0005 %
0006 % CALL:      obj = obj.setPoles(pz(2));
0007 %           obj = setPoles(obj, pz(2));
0008 %
0009 % INPUTS:    obj - is a miir object
0010 %
0011 % VERSION:    $Id: setPoles.m,v 1.1 2008/07/22 10:24:01 ingo Exp $
0012 %
0013 % HISTORY:    16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function obj = setPoles(obj, val)
0019     %% decide whether we modify the pzmodel-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'poles'
0023     obj.poles = val;
0024 end
0025
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setZeros

PURPOSE ^

SETZEROS Set the property 'zeros'

SYNOPSIS ^

function obj = setZeros(obj, val)

DESCRIPTION ^

SETZEROS Set the property 'zeros'

DESCRIPTION: SETZEROS Set the property 'zeros'

CALL:
obj = obj.setZeros(pz(-2));
obj = setZeros(obj, pz(-2));

INPUTS:
obj - is a miir object

VERSION:
\$Id: setZeros.m,v 1.1 2008/07/22 10:24:01 ingo Exp \$

HISTORY:
16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

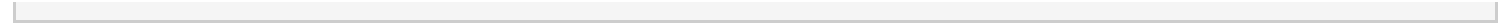
 [copy](#) COPY Make copy of pzmodel objects depending of the second input

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SOURCE CODE ^

```
0001 % SETZEROS Set the property 'zeros'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETZEROS Set the property 'zeros'
0005 %
0006 % CALL:      obj = obj.setZeros(pz(-2));
0007 %           obj = setZeros(obj, pz(-2));
0008 %
0009 % INPUTS:    obj - is a miir object
0010 %
0011 % VERSION:   $Id: setZeros.m,v 1.1 2008/07/22 10:24:01 ingo Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function obj = setZeros(obj, val)
0019     %% decide whether we modify the pzmodel-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'zeros'
0023     obj.zeros = val;
0024 end
0025
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [pzmodel](#) PZMODEL constructor for pzmodel class.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034         if isempty(obj_struct.plist)
0035             obj_struct.plist = plist();
0036         end
0037         ii = minfo('update_struct', 'pzmodel', '', '', ...
0038             '$Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $', ...
0039             {'Default'}, plist(), 0, 0);
0040         obj_struct.hist = history(time().utc_epoch_milli, ii, obj_struct.plist);
0041         obj_struct = rmfield(obj_struct, 'plist');
0042         obj_struct.creator = provenance('Updater 1.0 -> 1.1');
0043     catch ME
0044         disp(varargin{1});
0045         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (pzmodel) above is not
0046 from version 1.0')));
0047     end
0048     % Set the struct_ver to '1.1'
0049     struct_ver = '1.1';
0050 end
0051
0052 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.1 -> 1.2' %%%%%%%%%%%%%%%%%%%%%%%%%%
0053
0054 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0055     strcmp(struct_ver, '1.1')
0056     % Set the struct_ver to '1.2'
0057     struct_ver = '1.1';
0058 end
0059
0060 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.2 -> 1.3' %%%%%%%%%%%%%%%%%%%%%%%%%%
0061
0062 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0063     strcmp(struct_ver, '1.2')
0064     % Set the struct_ver to '1.3'
0065     struct_ver = '1.3';
0066 end
0067
0068 varargout{1} = obj_struct;
0069
0070

```

Index for m/sigproc/time_domain

Matlab files in this directory:

 Contents	%%%%%%%%%%%%%% path: m/sigproc/time_domain %%%%%%%%%%%%%%
 ltpda_gapfilling	LTPDA_GAPFILLING fills possible gaps in data.
 ltpda_polydetrend	LTPDA_POLYDETREND detrends the input analysis object using a polynomial of degree N.
 ltpda_spikecleaning	LTPDA_spikecleaning detects and corrects possible spikes in analysis objects
 ltpda_xcorr	LTPDA_XCORR makes cross-correlation estimates of the time-series

Subsequent directories:

 [CVS](#)

Index for m/sigproc/frequency_domain

Matlab files in this directory:

-  [Contents](#) %%%%%%%%%%%%%%% path: m/sigproc/frequency_domain %%%%%%%%%%%%%%%
-  [ltpda_cohere](#) LTPDA_COHERE makes coherence estimates of the time-series objects
-  [ltpda_cpsd](#) LTPDA_CPSD makes cross-spectral density estimates of the time-series objects.
-  [ltpda_lcohere](#) LTPDA_LCOHERE implement weighted coherence estimation computed on a log frequency axis.
-  [ltpda_lcpsd](#) LTPDA_LCPSD implement weighted cross-power estimation computed on a log frequency axis.
-  [ltpda_linedetect](#) LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
-  [ltpda_lpsd](#) LTPDA_LPSD implement LPSD algorithm for analysis objects.
-  [ltpda_ltf](#) LTPDA_LTFE implement transfer-function estimation computed on a log frequency axis.
-  [ltpda_nfest](#) LTPDA_NFEST Estimate the noise floor of the given frequency-series AO.
-  [ltpda_pwelch](#) LTPDA_PWELCH makes power spectral density estimates of the time-series objects
-  [ltpda_rms](#) LTPDA_RMS Calculate RMS deviation from spectrum
-  [ltpda_tfe](#) LTPDA_TFE makes transfer function estimates of the time-series objects.
-  [phasetrack](#)

Subsequent directories:

-  [CVS](#)

get_window

PURPOSE ^

GET_WINDOW returns the required window function as a structure.

SYNOPSIS ^

function w = get_window(varargin)

DESCRIPTION ^

GET_WINDOW returns the required window function as a structure.

DESCRIPTION: GET_WINDOW returns the required window function as a structure.

CALL: objs = copy(objs, [1,0]);

VERSION: \$Id: get_window.m,v 1.2 2008/07/14 10:40:13 ingo Exp \$

HISTORY: 18-05-2007 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 specwin % SPECWIN spectral window object class constructor.

SUBFUNCTIONS ^

 function w = get_kaiser_window(w, N, psll)

SOURCE CODE ^

```
0001 % GET_WINDOW returns the required window function as a structure.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: GET_WINDOW returns the required window function as a structure.
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % VERSION:   $Id: get_window.m,v 1.2 2008/07/14 10:40:13 ingo Exp $
0009 %
0010 % HISTORY:   18-05-2007 M Hewitson
0011 %              Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014 %
0015 function w = get_window(varargin)
0016
0017     PSLL = [];
0018
0019     % Get inputs
0020     if nargin == 3
0021         w = varargin{1};
0022         name = varargin{2};
0023         N = varargin{3};
0024     elseif nargin == 4
```

```

0025     w     = varargin{1};
0026     name  = varargin{2};
0027     N     = varargin{3};
0028     PSLL  = varargin{4};
0029     else
0030         error('### Incorrect inputs. ');
0031     end
0032
0033     % Get window
0034     if isempty(PSLL)
0035         % get standard window
0036         z = [0:N-1]./N;
0037         win_name = sprintf('win_%s', name);
0038         try
0039             w = feval(win_name, w, z);
0040         catch
0041             error('\n### Your window [%s] is not a supported window.\n### Please type >>
specwin.getTypes to see all supported windows.', win_name(5:end));
0042         end
0043     else
0044
0045         % Deal with Kaiser
0046         w = get_kaiser_window(w, N, PSLL);
0047     end
0048 end
0049
0050
0051
0052 %-----
0053 % Make Kaiser window
0054 %
0055 function w = get_kaiser_window(w, N, psll)
0056
0057     % Make window struct
0058     w.type      = 'Kaiser';
0059     w.alpha     = specwin.kaiser_alpha(psll);
0060     w.psll      = psll;
0061     w.rov       = specwin.kaiser_rov(w.alpha);
0062     w.nenbw     = specwin.kaiser_nenbw(w.alpha);
0063     w.w3db      = specwin.kaiser_w3db(w.alpha);
0064     w.flatness  = specwin.kaiser_flatness(w.alpha);
0065
0066     % Compute window samples
0067     % - here we adjust to make the window asymmetric
0068     v = kaiser(N+1, pi*w.alpha);
0069     v = v(1:end-1);
0070     w.ws        = sum(v);
0071     w.ws2       = sum(v.*v);
0072     w.win       = v.';
0073 end

```

[Home](#) > [classes](#) > [@specwin](#) > kaiser_alpha.m

kaiser_alpha

PURPOSE [^]

KAISER_ALPHA returns the alpha parameter that gives the required input

SYNOPSIS [^]

```
function alpha = kaiser_alpha(req_psl1)
```

DESCRIPTION [^]

KAISER_ALPHA returns the alpha parameter that gives the required input PSLL.

Taken from C code of Gerhard Heinzel:

Compute the parameter alpha of Kaiser windows from the required PSLL [dB]. Best-fit polynomial was obtained from 180 data points between alpha=1 and alpha=9.95. Maximum error is 0.05
Maximum error for PSLL > 30 dB is 0.02

M Hewitson 19-05-07

\$Id: kaiser_alpha.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```

0001 function alpha = kaiser_alpha(req_psl1)
0002
0003 % KAISER_ALPHA returns the alpha parameter that gives the required input
0004 % PSLL.
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the parameter alpha of Kaiser windows
0009 %     from the required PSLL [dB]. Best-fit polynomial
0010 %     was obtained from 180 data points between alpha=1
0011 %     and alpha=9.95. Maximum error is 0.05
0012 %     Maximum error for PSLL > 30 dB is 0.02
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_alpha.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0017 %
0018 %
0019 %
0020 a0 = -0.0821377;
0021 a1 = 4.71469;
0022 a2 = -0.493285;
0023 a3 = 0.0889732;
0024
0025 x = req_psl1 / 100;
0026 alpha = (((((a3 * x) + a2) * x) + a1) * x + a0);
0027
0028
0029 % END

```


[Home](#) > [classes](#) > [@specwin](#) > kaiser_flatness.m

kaiser_flatness

PURPOSE [^]

KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser

SYNOPSIS [^]

```
function flatness = kaiser_flatness(alpha)
```

DESCRIPTION [^]

KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser window with parameter alpha.

Taken from C code of Gerhard Heinzel:

Compute the flatness in the central bin [dB]
of Kaiser windows from the parameter alpha.
Best-fit polynomial was obtained from 180 data
points between alpha=1 and alpha=9.95.
Maximum error is 0.013 dB.

M Hewitson 19-05-07

\$Id: kaiser_flatness.m,v 1.1 2008/06/20 10:46:55 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```

0001 function flatness = kaiser_flatness(alpha)
0002
0003 % KAISER_FLATNESS returns the flatness in dB of the central bin of a kaiser
0004 % window with parameter alpha.
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the flatness in the central bin [dB]
0009 %     of Kaiser windows from the parameter alpha.
0010 %     Best-fit polynomial was obtained from 180 data
0011 %     points between alpha=1 and alpha=9.95.
0012 %     Maximum error is 0.013 dB.
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_flatness.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0017 %
0018
0019 a0      = 0.141273;
0020 a1      = 0.262425;
0021 a2      = 0.00642551;
0022 a3      = -0.000405621;
0023 x       = alpha;
0024 flatness = -1. / (((((a3 * x) + a2) * x) + a1) * x + a0);
0025
0026 % END

```


[Home](#) > [classes](#) > [@specwin](#) > kaiser_nenbw.m

kaiser_nenbw

PURPOSE [^]

KAISER_NENBW returns the normalized noise-equivalent bandwidth for a

SYNOPSIS [^]

```
function nenbw = kaiser_nenbw(alpha)
```

DESCRIPTION [^]

KAISER_NENBW returns the normalized noise-equivalent bandwidth for a kaiser window with parameter alpha.

Take from C code of Gerhard Heinzel:

```
Compute the 'normalized noise-equivalent bandwidth'
(NENBW) [bins] of Kaiser windows from the parameter alpha.
Best-fit polynomial was obtained from 180 data
points between alpha=1 and alpha=9.95.
Maximum error is 0.007 bins
NOTE that NENBW can be computed precisely
from the actual time-domain window values
```

M Hewitson 19-05-07

\$Id: kaiser_nenbw.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

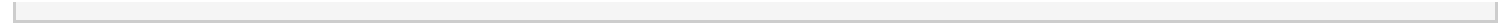
This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function nenbw = kaiser_nenbw(alpha)
0002
0003 % KAISER_NENBW returns the normalized noise-equivalent bandwidth for a
0004 % kaiser window with parameter alpha.
0005 %
0006 % Take from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the 'normalized noise-equivalent bandwidth'
0009 %     (NENBW) [bins] of Kaiser windows from the parameter alpha.
0010 %     Best-fit polynomial was obtained from 180 data
0011 %     points between alpha=1 and alpha=9.95.
0012 %     Maximum error is 0.007 bins
0013 %     NOTE that NENBW can be computed precisely
0014 %     from the actual time-domain window values
0015 %
0016 % M Hewitson 19-05-07
0017 %
0018 % $Id: kaiser_nenbw.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0019 %
0020
0021 a0    = 0.768049;
0022 a1    = 0.411986;
0023 a2    = -0.0264817;
0024 a3    = 0.000962211;
0025 x     = alpha;
0026 nenbw = (((((a3 * x) + a2) * x) + a1) * x + a0);
0027
0028 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

kaiser_rov

PURPOSE ^

KAISER_ROV returns the recommended overlap for a Kaiser window with

SYNOPSIS ^

function rov = kaiser_rov(alpha)

DESCRIPTION ^

KAISER_ROV returns the recommended overlap for a Kaiser window with parameter alpha.

Taken from C code of Gerhard Heinzel:

Compute the 'recommended overlap' (ROV) [%] of Kaiser windows from the parameter alpha. Best-fit polynomial was obtained from 180 data points between alpha=1 and alpha=9.95. Maximum error is 1.5%, mainly due to insufficient precision in the data points

M Hewitson 19-05-07

\$Id: kaiser_rov.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function rov = kaiser_rov(alpha)
0002
0003 % KAISER_ROV returns the recommended overlap for a Kaiser window with
0004 % parameter alpha.
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the 'recommended overlap' (ROV) [%] of Kaiser windows
0009 %     from the parameter alpha. Best-fit polynomial
0010 %     was obtained from 180 data points between alpha=1
0011 %     and alpha=9.95. Maximum error is 1.5%, mainly due
0012 %     to insufficient precision in the data points
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_rov.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0017 %
0018 %
0019 %
0020 a0 = 0.0061076;
0021 a1 = 0.00912223;
0022 a2 = -0.000925946;
0023 a3 = 4.42204e-05;
0024 x = alpha;
0025 rov = 100 - 1 / (((((a3 * x) + a2) * x) + a1) * x + a0);
0026
0027 % END
```


kaiser_w3db

PURPOSE ^

KAISER_W3DB returns the 3dB bandwidth in bins of a kaiser window with

SYNOPSIS ^

function w3db = kaiser_w3db(alpha)

DESCRIPTION ^

KAISER_W3DB returns the 3dB bandwidth in bins of a kaiser window with parameter alpha.

Taken from C code of Gerhard Heinzel:

Compute the 3dB bandwidth (w3db) [bins]
of Kaiser windows from the parameter alpha.
Best-fit polynomial was obtained from 180 data
points between alpha=1 and alpha=9.95.
Maximum error is 0.006 bins.

M Hewitson 19-05-07

\$Id: kaiser_w3db.m,v 1.1 2008/06/20 10:46:55 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w3db = kaiser_w3db(alpha)
0002
0003 % KAISER_W3DB returns the 3dB bandwidth in bins of a kaiser window with
0004 % parameter alpha.
0005 %
0006 % Taken from C code of Gerhard Heinzel:
0007 %
0008 %     Compute the 3dB bandwidth (w3db) [bins]
0009 %     of Kaiser windows from the parameter alpha.
0010 %     Best-fit polynomial was obtained from 180 data
0011 %     points between alpha=1 and alpha=9.95.
0012 %     Maximum error is 0.006 bins.
0013 %
0014 % M Hewitson 19-05-07
0015 %
0016 % $Id: kaiser_w3db.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0017 %
0018
0019 a0 = 0.757185;
0020 a1 = 0.377847;
0021 a2 = -0.0238342;
0022 a3 = 0.00086012;
0023 x = alpha;
0024 w3db = (((((a3 * x) + a2) * x) + a1) * x + a0);
0025
0026 % END
```


update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035         obj_struct.type = obj_struct.name;
0036         obj_struct = rmfield(obj_struct, 'name');
0037         obj_struct = rmfield(obj_struct, 'created');
0038         obj_struct = rmfield(obj_struct, 'plist');
0039     catch ME
0040         disp(varargin{1});
0041         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (specwin) above is not
from version 1.0')));
0042     end
0043
0044     % Set the struct_ver to '1.1'
0045     struct_ver = '1.1';
0046
0047 end
0048
0049 %%%%%%%%%% Update version '1.1 -> 1.2' %%%%%%%%%%
0050
0051 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0052     strcmp(struct_ver, '1.1')
0053
0054     % Set the struct_ver to '1.2'
0055     struct_ver = '1.1';
0056
0057 end
0058
0059 %%%%%%%%%% Update version '1.2 -> 1.3' %%%%%%%%%%
0060
0061 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0062     strcmp(struct_ver, '1.2')
0063
0064     % Set the struct_ver to '1.3'
0065     struct_ver = '1.3';
0066
0067 end
0068
0069 varargout{1} = obj_struct;
0070

```

[Home](#) > [classes](#) > [@specwin](#) > win_BH92.m

win_BH92

PURPOSE [^]

WIN_BH92 return BH92 window sample for input z.

SYNOPSIS [^]

```
function w = win_BH92(w, z)
```

DESCRIPTION [^]

```
WIN_BH92 return BH92 window sample for input z.
M Hewitson 18-05-07
$Id: win_BH92.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_BH92(w, z)
0002
0003 % WIN_BH92 return BH92 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_BH92.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.35875 - 0.48829 * cos (z) + 0.14128 * cos (2 * z) - 0.01168 * cos (3 * z);
0012
0013 % Make window struct
0014 w.type      = 'BH92';
0015 w.alpha     = 0;
0016 w.psll      = 92;
0017 w.rov       = 66.1;
0018 w.nenbw     = 2.0044;
0019 w.w3db      = 1.8962;
0020 w.flatness  = -0.8256;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

win_Bartlett

PURPOSE ^

WIN_BARTLETT return Bartlett window sample for input z.

SYNOPSIS ^

function w = win_Bartlett(w, z)

DESCRIPTION ^

WIN_BARTLETT return Bartlett window sample for input z.
M Hewitson 18-05-07
\$Id: win_Bartlett.m,v 1.1 2008/06/20 10:46:55 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Bartlett(w, z)
0002
0003 % WIN_BARTLETT return Bartlett window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Bartlett.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0008 %
0009 %
0010
0011 v = z * 2;
0012 idx = find(v>1);
0013 v(idx) = 2-v(idx);
0014
0015
0016 % Make window struct
0017 w.type      = 'Bartlett';
0018 w.alpha     = 0;
0019 w.psll      = 26.5;
0020 w.rov       = 50;
0021 w.nenbw     = 1.3333;
0022 w.w3db      = 1.2736;
0023 w.flatness  = -1.8242;
0024 w.ws        = sum(v);
0025 w.ws2       = sum(v.*v);
0026 w.win       = v;
0027
0028
0029
0030 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_FTHP.m

win_FTHP

PURPOSE [^]

WIN_FTHP return FTHP window sample for input z.

SYNOPSIS [^]

```
function w = win_FTHP(w, z)
```

DESCRIPTION [^]

```
WIN_FTHP return FTHP window sample for input z.
M Hewitson 18-05-07
$Id: win_FTHP.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_FTHP(w, z)
0002
0003 % WIN_FTHP return FTHP window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_FTHP.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = pi * (2 * z - 1);
0011 v = (1 + 1.912510941 * cos(z) + 1.079173272 * cos(2 * z) + ...
0012      0.1832630879 * cos(3 * z));
0013
0014
0015 % Make window struct
0016 w.type      = 'FTHP';
0017 w.alpha     = 0;
0018 w.psll      = 70.4;
0019 w.rov       = 72.3;
0020 w.nenbw     = 3.4279;
0021 w.w3db      = 3.3846;
0022 w.flatness  = 0.0096;
0023 w.ws        = sum(v);
0024 w.ws2       = sum(v.*v);
0025 w.win       = v;
0026
0027 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_FTNI.m

win_FTNI

PURPOSE ^

WIN_FTNI return FTNI window sample for input z.

SYNOPSIS ^

```
function w = win_FTNI(w, z)
```

DESCRIPTION ^

```
WIN_FTNI return FTNI window sample for input z.
M Hewitson 18-05-07
$Id: win_FTNI.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_FTNI(w, z)
0002
0003 % WIN_FTNI return FTNI window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_FTNI.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.2810639 - 0.5208972 * cos (z) + 0.1980399 * cos (2 * z);
0012
0013 % Make window struct
0014 w.type      = 'FTNI';
0015 w.alpha     = 0;
0016 w.psll      = 44.4;
0017 w.rov       = 65.6;
0018 w.nenbw     = 2.9656;
0019 w.w3db      = 2.9355;
0020 w.flatness  = 0.0169;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

win_FTSRS

PURPOSE ^

WIN_FTSRS return FTSRS window sample for input z.

SYNOPSIS ^

function w = win_FTSRS(w, z)

DESCRIPTION ^

WIN_FTSRS return FTSRS window sample for input z.
M Hewitson 18-05-07
\$Id: win_FTSRS.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_FTSRS(w, z)
0002
0003 % WIN_FTSRS return FTSRS window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_FTSRS.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009 %
0010 z = z * 2 * pi;
0011 v = (1.0 - 1.93 * cos (z) + 1.29 ...
0012      * cos (2 * z) - 0.388 * cos (3 * z) + 0.028 * cos (4 * z));
0013
0014 % Make window struct
0015 w.type      = 'FTSRS';
0016 w.alpha     = 0;
0017 w.psll      = 76.6;
0018 w.rov       = 75.4;
0019 w.nenbw     = 3.7702;
0020 w.w3db      = 3.7274;
0021 w.flatness  = -0.0156;
0022 w.ws        = sum(v);
0023 w.ws2       = sum(v.*v);
0024 w.win       = v;
0025
0026 % END
```


win_HFT116D

PURPOSE ^

WIN_HFT116D return HFT116D window sample for input z.

SYNOPSIS ^

function w = win_HFT116D(w, z)

DESCRIPTION ^

```
WIN_HFT116D return HFT116D window sample for input z.  
M Hewitson 18-05-07  
$Id: win_HFT116D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_HFT116D(w, z)  
0002  
0003 % WIN_HFT116D return HFT116D window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_HFT116D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $  
0008 %  
0009  
0010 z = z * 2 * pi;  
0011 v = (1 - 1.9575375 * cos (z) + ...  
0012      1.4780705 * cos (2 * z) - 0.6367431 * cos (3 * z) + ...  
0013      0.1228389 * cos (4 * z) - 0.0066288 * cos (5 * z));  
0014  
0015 % Make window struct  
0016 w.type      = 'HFT116D';  
0017 w.alpha     = 0;  
0018 w.psll      = 116.8;  
0019 w.rov       = 78.2;  
0020 w.nenbw     = 4.2186;  
0021 w.w3db      = 4.1579;  
0022 w.flatness  = -0.0028;  
0023 w.ws        = sum(v);  
0024 w.ws2       = sum(v.*v);  
0025 w.win       = v;  
0026  
0027 % END
```


win_HFT144D

PURPOSE ^

WIN_HFT144D return HFT144D window sample for input z.

SYNOPSIS ^

function w = win_HFT144D(w, z)

DESCRIPTION ^

WIN_HFT144D return HFT144D window sample for input z.
M Hewitson 18-05-07
\$Id: win_HFT144D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_HFT144D(w, z)
0002
0003 % WIN_HFT144D return HFT144D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT144D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.96760033 * cos (z) ...
0012      + 1.57983607 * cos (2 * z) - 0.81123644 * cos (3 * z) + ...
0013      0.22583558 * cos (4 * z) - 0.02773848 * cos (5 * z) + ...
0014      0.00090360 * cos (6 * z));
0015
0016 % Make window struct
0017 w.type      = 'HFT144D';
0018 w.alpha     = 0;
0019 w.psll      = 144.1;
0020 w.rov       = 79.9;
0021 w.nenbw     = 4.5386;
0022 w.w3db      = 4.4697;
0023 w.flatness  = 0.0021;
0024 w.ws        = sum(v);
0025 w.ws2       = sum(v.*v);
0026 w.win       = v;
0027
0028 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_HFT169D.m

win_HFT169D

PURPOSE [^]

WIN_HFT169D return HFT169D window sample for input z.

SYNOPSIS [^]

```
function w = win_HFT169D(w, z)
```

DESCRIPTION [^]

```
WIN_HFT169D return HFT169D window sample for input z.  
M Hewitson 18-05-07  
$Id: win_HFT169D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_HFT169D(w, z)
0002
0003 % WIN_HFT169D return HFT169D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT169D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.97441843 * cos (z) ...
0012      + 1.65409889 * cos (2 * z) - 0.95788187 * cos (3 * z) + ...
0013      0.33673420 * cos (4 * z) - 0.06364622 * cos (5 * z) + ...
0014      0.00521942 * cos (6 * z) - 0.00010599 * cos (7 * z));
0015
0016 % Make window struct
0017 w.type      = 'HFT169D';
0018 w.alpha     = 0;
0019 w.psll      = 169.5;
0020 w.rov       = 81.2;
0021 w.nenbw     = 4.8347;
0022 w.w3db      = 4.7588;
0023 w.flatness  = 0.0017;
0024 w.ws        = sum(v);
0025 w.ws2       = sum(v.*v);
0026 w.win       = v;
0027
0028 % END
```


win_HFT196D

PURPOSE ^

WIN_HFT196D return HFT196D window sample for input z.

SYNOPSIS ^

function w = win_HFT196D(w, z)

DESCRIPTION ^

WIN_HFT196D return HFT196D window sample for input z.
M Hewitson 18-05-07
\$Id: win_HFT196D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_HFT196D(w, z)
0002
0003 % WIN_HFT196D return HFT196D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT196D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.979280420 * cos (z) ...
0012      + 1.710288951 * cos (2 * z) - 1.081629853 * cos (3 * z) + ...
0013      0.448734314 * cos (4 * z) - 0.112376628 * cos (5 * z) + ...
0014      0.015122992 * cos (6 * z) - 0.000871252 * cos (7 * z) + ...
0015      0.000011896 * cos (8 * z));
0016
0017 % Make window struct
0018 w.type      = 'HFT196D';
0019 w.alpha     = 0;
0020 w.psll      = 196.2;
0021 w.rov       = 82.3;
0022 w.nenbw     = 5.1134;
0023 w.w3db      = 5.0308;
0024 w.flatness  = 0.0013;
0025 w.ws        = sum(v);
0026 w.ws2       = sum(v.*v);
0027 w.win       = v;
0028
0029 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_HFT223D.m

win_HFT223D

PURPOSE [^]

WIN_HFT223D return HFT223D window sample for input z.

SYNOPSIS [^]

```
function w = win_HFT223D(w, z)
```

DESCRIPTION [^]

```
WIN_HFT223D return HFT223D window sample for input z.
M Hewitson 18-05-07
$Id: win_HFT223D.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_HFT223D(w, z)
0002
0003 % WIN_HFT223D return HFT223D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT223D.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.98298997309 * cos(z) + ...
0012      1.75556083063 * cos (2 * z) - 1.19037717712 * cos (3 * z) + ...
0013      0.56155440797 * cos (4 * z) - 0.17296769663 * cos (5 * z) + ...
0014      0.03233247087 * cos (6 * z) - 0.00324954578 * cos (7 * z) + ...
0015      0.00013801040 * cos (8 * z) - 0.00000132725 * cos (9 * z));
0016
0017 % Make window struct
0018 w.type = 'HFT223D';
0019 w.alpha = 0;
0020 w.psl1 = 223;
0021 w.rov = 83.3;
0022 w.nenbw = 5.3888;
0023 w.w3db = 5.3;
0024 w.flatness = -0.0011;
0025 w.ws = sum(v);
0026 w.ws2 = sum(v.*v);
0027 w.win = v;
0028
0029 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

win_HFT248D

PURPOSE [^]

WIN_HFT248D return HFT248D window sample for input z.

SYNOPSIS [^]

function w = win_HFT248D(w, z)

DESCRIPTION [^]

WIN_HFT248D return HFT248D window sample for input z.
M Hewitson 18-05-07
\$Id: win_HFT248D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_HFT248D(w, z)
0002
0003 % WIN_HFT248D return HFT248D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT248D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.985844164102 * cos(z) + ...
0012      1.791176438506 * cos(2 * z) - 1.282075284005 * cos(3 * z) + ...
0013      0.667777530266 * cos(4 * z) - 0.240160796576 * cos(5 * z) + ...
0014      0.056656381764 * cos(6 * z) - 0.008134974479 * cos(7 * z) + ...
0015      0.000624544650 * cos(8 * z) - 0.000019808998 * cos(9 * z) + ...
0016      0.000000132974 * cos(10 * z));
0017
0018 % Make window struct
0019 w.type      = 'HFT248D';
0020 w.alpha     = 0;
0021 w.psll      = 248.4;
0022 w.rov       = 84.1;
0023 w.nenbw     = 5.6512;
0024 w.w3db      = 5.5567;
0025 w.flatness  = 0.0009;
0026 w.ws        = sum(v);
0027 w.ws2       = sum(v.*v);
0028 w.win       = v;
0029
0030 % END
```


win_HFT70

PURPOSE ^

WIN_HFT70 return HFT70 window sample for input z.

SYNOPSIS ^

function w = win_HFT70(w, z)

DESCRIPTION ^

```
WIN_HFT70 return HFT70 window sample for input z.  
M Hewitson 18-05-07  
$Id: win_HFT70.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_HFT70(w, z)  
0002  
0003 % WIN_HFT70 return HFT70 window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_HFT70.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $  
0008 %  
0009  
0010 z = z * 2 * pi;  
0011 v = (1 - 1.90796 * cos (z) + ...  
0012      1.07349 * cos (2 * z) - 0.18199 * cos (3 * z));  
0013  
0014 % Make window struct  
0015 w.type      = 'HFT70';  
0016 w.alpha     = 0;  
0017 w.psll      = 70.4;  
0018 w.rov       = 72.2;  
0019 w.nenbw     = 3.4129;  
0020 w.w3db      = 3.3720;  
0021 w.flatness  = -0.0065;  
0022 w.ws        = sum(v);  
0023 w.ws2       = sum(v.*v);  
0024 w.win       = v;  
0025  
0026 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_HFT90D.m

win_HFT90D

PURPOSE [^]

WIN_HFT90D return HFT90D window sample for input z.

SYNOPSIS [^]

```
function w = win_HFT90D(w, z)
```

DESCRIPTION [^]

```
WIN_HFT90D return HFT90D window sample for input z.  
M Hewitson 18-05-07  
$Id: win_HFT90D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_HFT90D(w, z)
0002
0003 % WIN_HFT90D return HFT90D window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_HFT90D.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = (1 - 1.942604 * cos (z) + ...
0012      1.340318 * cos (2 * z) - 0.440811 * cos (3 * z) + ...
0013      0.043097 * cos (4 * z));
0014
0015 % Make window struct
0016 w.type      = 'HFT90D';
0017 w.alpha     = 0;
0018 w.psll      = 90.2;
0019 w.rov       = 76;
0020 w.nenbw     = 3.8832;
0021 w.w3db      = 3.8320;
0022 w.flatness  = -0.0039;
0023 w.ws        = sum(v);
0024 w.ws2       = sum(v.*v);
0025 w.win       = v;
0026
0027 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

win_HFT95

PURPOSE ^

WIN_HFT95 return HFT95 window sample for input z.

SYNOPSIS ^

function w = win_HFT95(w, z)

DESCRIPTION ^

```
WIN_HFT95 return HFT95 window sample for input z.  
M Hewitson 18-05-07  
$Id: win_HFT95.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_HFT95(w, z)  
0002  
0003 % WIN_HFT95 return HFT95 window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_HFT95.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $  
0008 %  
0009  
0010 z = z * 2 * pi;  
0011 v = (1 - 1.9383379 * cos (z) + ...  
0012      1.3045202 * cos (2 * z) - 0.4028270 * cos (3 * z) + ...  
0013      0.0350665 * cos (4 * z));  
0014  
0015 % Make window struct  
0016 w.type      = 'HFT95';  
0017 w.alpha     = 0;  
0018 w.psll      = 95;  
0019 w.rov       = 75.6;  
0020 w.nenbw     = 3.8112;  
0021 w.w3db      = 3.759;  
0022 w.flatness  = 0.0044;  
0023 w.ws        = sum(v);  
0024 w.ws2       = sum(v.*v);  
0025 w.win       = v;  
0026  
0027 % END
```


win_Hamming

PURPOSE ^

WIN_HAMMING return Hamming window sample for input z.

SYNOPSIS ^

function w = win_Hamming(w, z)

DESCRIPTION ^

WIN_HAMMING return Hamming window sample for input z.
M Hewitson 18-05-07
\$Id: win_Hamming.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Hamming(w, z)
0002
0003 % WIN_HAMMING return Hamming window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Hamming.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 v = 0.54 - 0.46 * cos(2*pi*z);
0011
0012 % Make window struct
0013 w.type      = 'Hamming';
0014 w.alpha     = 0;
0015 w.psll      = 42.7;
0016 w.rov       = 50;
0017 w.nenbw     = 1.3628;
0018 w.w3db      = 1.3008;
0019 w.flatness  = -1.7514;
0020 w.ws        = sum(v);
0021 w.ws2       = sum(v.*v);
0022 w.win       = v;
0023
0024 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_Hanning.m

win_Hanning

PURPOSE [^]

WIN_HANNING return Hanning window sample for input z.

SYNOPSIS [^]

```
function w = win_Hanning(w, z)
```

DESCRIPTION [^]

```
WIN_HANNING return Hanning window sample for input z.
M Hewitson 18-05-07
$Id: win_Hanning.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_Hanning(w, z)
0002
0003 % WIN_HANNING return Hanning window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Hanning.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0008 %
0009
0010 v = 0.5 * (1 - cos(2*pi*z));
0011
0012 % Make window struct
0013 w.type      = 'Hanning';
0014 w.alpha     = 0;
0015 w.psll      = 31.5;
0016 w.rov       = 50;
0017 w.nenbw     = 1.5;
0018 w.w3db      = 1.4382;
0019 w.flatness  = -1.4236;
0020 w.ws        = sum(v);
0021 w.ws2       = sum(v.*v);
0022 w.win       = v;
0023
0024 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

win_Nuttall3

PURPOSE ^

WIN_NUTTALL3 return Nuttall3 window sample for input z.

SYNOPSIS ^

function w = win_Nuttall3(w, z)

DESCRIPTION ^

WIN_NUTTALL3 return Nuttall3 window sample for input z.
M Hewitson 18-05-07
\$Id: win_Nuttall3.m,v 1.1 2008/06/20 10:46:55 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Nuttall3(w, z)
0002
0003 % WIN_NUTTALL3 return Nuttall3 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall3.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.375 - 0.5 * cos (z) + 0.125 * cos (2 * z);
0012
0013 % Make window struct
0014 w.type      = 'Nuttall3';
0015 w.alpha     = 0;
0016 w.psll      = 46.7;
0017 w.rov       = 64.7;
0018 w.nenbw     = 1.9444;
0019 w.w3db      = 1.8496;
0020 w.flatness  = -0.8630;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```


win_Nuttall3a

PURPOSE ^

WIN_NUTTALL3A return Nuttall3a window sample for input z.

SYNOPSIS ^

function w = win_Nuttall3a(w, z)

DESCRIPTION ^

WIN_NUTTALL3A return Nuttall3a window sample for input z.
M Hewitson 18-05-07
\$Id: win_Nuttall3a.m,v 1.1 2008/06/20 10:46:55 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Nuttall3a(w, z)
0002
0003 % WIN_NUTTALL3A return Nuttall3a window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall3a.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.40897 - 0.5 * cos (z) + 0.09103 * cos (2 * z);
0012
0013 % Make window struct
0014 w.type      = 'Nuttall3a';
0015 w.alpha     = 0;
0016 w.psll      = 64.2;
0017 w.rov       = 61.2;
0018 w.nenbw     = 1.7721;
0019 w.w3db      = 1.6828;
0020 w.flatness  = -1.0453;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```


win_Nuttall3b

PURPOSE ^

WIN_NUTTALL3B return Nuttall3b window sample for input z.

SYNOPSIS ^

function w = win_Nuttall3b(w, z)

DESCRIPTION ^

WIN_NUTTALL3B return Nuttall3b window sample for input z.
M Hewitson 18-05-07
\$Id: win_Nuttall3b.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Nuttall3b(w, z)
0002
0003 % WIN_NUTTALL3B return Nuttall3b window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall3b.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.4243801 - 0.4973406 * cos (z) + 0.0782793 * cos (2 * z);
0012
0013 % Make window struct
0014 w.type      = 'Nuttall3b';
0015 w.alpha     = 0;
0016 w.psll      = 71.5;
0017 w.rov       = 59.8;
0018 w.nenbw     = 1.7037;
0019 w.w3db      = 1.6162;
0020 w.flatness  = -1.1352;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_Nuttall4.m

win_Nuttall4

PURPOSE [^]

WIN_NUTTALL4 return Nuttall4 window sample for input z.

SYNOPSIS [^]

```
function w = win_Nuttall4(w, z)
```

DESCRIPTION [^]

```
WIN_NUTTALL4 return Nuttall4 window sample for input z.
M Hewitson 18-05-07
$Id: win_Nuttall4.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_Nuttall4(w, z)
0002
0003 % WIN_NUTTALL4 return Nuttall4 window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009 %
0010 z = z * 2 * pi;
0011 v = 0.3125 - 0.46875 * cos(z) + 0.1875 * cos (2 * z) - 0.03125 * cos (3 * z);
0012
0013 % Make window struct
0014 w.type      = 'Nuttall4';
0015 w.alpha     = 0;
0016 w.psll      = 60.9;
0017 w.rov       = 70.5;
0018 w.nenbw     = 2.31;
0019 w.w3db      = 2.1884;
0020 w.flatness  = -0.6184;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

win_Nuttall4a

PURPOSE ^

WIN_NUTTALL4A return Nuttall4a window sample for input z.

SYNOPSIS ^

function w = win_Nuttall4a(w, z)

DESCRIPTION ^

WIN_NUTTALL4A return Nuttall4a window sample for input z.
M Hewitson 18-05-07
\$Id: win_Nuttall4a.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Nuttall4a(w, z)
0002
0003 % WIN_NUTTALL4A return Nuttall4a window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4a.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.338946 - 0.481973 * cos (z) + 0.161054 * cos (2 * z) - 0.018027 * cos (3 * z);
0012
0013 % Make window struct
0014 w.type      = 'Nuttall4a';
0015 w.alpha    = 0;
0016 w.psll     = 82.6;
0017 w.rov      = 68;
0018 w.nenbw    = 2.1253;
0019 w.w3db     = 2.0123;
0020 w.flatness  = -0.7321;
0021 w.ws       = sum(v);
0022 w.ws2      = sum(v.*v);
0023 w.win      = v;
0024
0025 % END
```


win_Nuttall4b

PURPOSE ^

WIN_NUTTALL4B return Nuttall4b window sample for input z.

SYNOPSIS ^

function w = win_Nuttall4b(w, z)

DESCRIPTION ^

WIN_NUTTALL4B return Nuttall4b window sample for input z.
M Hewitson 18-05-07
\$Id: win_Nuttall4b.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Nuttall4b(w, z)
0002
0003 % WIN_NUTTALL4B return Nuttall4b window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4b.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.355768 - 0.487396 * cos (z) + 0.144232 * cos (2 * z) - 0.012604 * cos (3 * z);
0012
0013 % Make window struct
0014 w.type      = 'Nuttall4b';
0015 w.alpha     = 0;
0016 w.psll      = 93.3;
0017 w.rov       = 66.3;
0018 w.nenbw     = 2.0212;
0019 w.w3db      = 1.9122;
0020 w.flatness  = -0.8118;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```


win_Nuttall4c

PURPOSE ^

WIN_NUTTALL4C return Nuttall4c window sample for input z.

SYNOPSIS ^

function w = win_Nuttall4c(w, z)

DESCRIPTION ^

WIN_NUTTALL4C return Nuttall4c window sample for input z.
M Hewitson 18-05-07
\$Id: win_Nuttall4c.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Nuttall4c(w, z)
0002
0003 % WIN_NUTTALL4C return Nuttall4c window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Nuttall4c.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.3635819 - 0.4891775 * cos (z) + 0.1365995 * cos (2 * z) - 0.0106411 * cos (3 *
z);
0012
0013 % Make window struct
0014 w.type      = 'Nuttall4c';
0015 w.alpha     = 0;
0016 w.psll      = 98.1;
0017 w.rov       = 65.6;
0018 w.nenbw     = 1.9761;
0019 w.w3db      = 1.8687;
0020 w.flatness  = -0.8506;
0021 w.ws        = sum(v);
0022 w.ws2       = sum(v.*v);
0023 w.win       = v;
0024
0025 % END
```


[Home](#) > [classes](#) > [@specwin](#) > win_Rectangular.m

win_Rectangular

PURPOSE ^

WIN_RECTANGULAR return rectangular window sample for input z.

SYNOPSIS ^

```
function w = win_Rectangular(w, z)
```

DESCRIPTION ^

```
WIN_RECTANGULAR return rectangular window sample for input z.
M Hewitson 18-05-07
$Id: win_Rectangular.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Rectangular(w, z)
0002
0003 % WIN_RECTANGULAR return rectangular window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_Rectangular.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 v = ones(size(z));
0011
0012 % Make window struct
0013 w.type      = 'Rectangular';
0014 w.alpha     = 0;
0015 w.psll      = 13.3;
0016 w.rov       = 0.0;
0017 w.nenbw     = 1.0;
0018 w.w3db      = 0.8845;
0019 w.flatness  = -3.9224;
0020 w.ws        = sum(v);
0021 w.ws2       = sum(v.*v);
0022 w.win       = v;
0023
0024 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

win_SFT3F

PURPOSE ^

WIN_SFT3F return SFT3F window sample for input z.

SYNOPSIS ^

function w = win_SFT3F(w, z)

DESCRIPTION ^

WIN_SFT3F return SFT3F window sample for input z.
M Hewitson 18-05-07
\$Id: win_SFT3F.m,v 1.1 2008/06/20 10:46:54 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_SFT3F(w, z)
0002
0003 % WIN_SFT3F return SFT3F window sample for input z.
0004 %
0005 % M Hewitson 18-05-07
0006 %
0007 % $Id: win_SFT3F.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
0008 %
0009
0010 z = z * 2 * pi;
0011 v = 0.26526 - 0.5 * cos (z) + 0.23474 * cos (2 * z);
0012
0013 % Make window struct
0014 w.type      = 'SFT3F';
0015 w.alpha    = 0;
0016 w.psll     = 31.7;
0017 w.rov      = 66.7;
0018 w.nenbw    = 3.1681;
0019 w.w3db     = 3.1502;
0020 w.flatness  = 0.0082;
0021 w.ws       = sum(v);
0022 w.ws2      = sum(v.*v);
0023 w.win      = v;
0024
0025 % END
```


win_SFT3M

PURPOSE ^

WIN_SFT3M return SFT3M window sample for input z.

SYNOPSIS ^

function w = win_SFT3M(w, z)

DESCRIPTION ^

```
WIN_SFT3M return SFT3M window sample for input z.  
M Hewitson 18-05-07  
$Id: win_SFT3M.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_SFT3M(w, z)  
0002  
0003 % WIN_SFT3M return SFT3M window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_SFT3M.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $  
0008 %  
0009  
0010 z = z * 2 * pi;  
0011 v = 0.28235 - 0.52105 * cos (z) + 0.19659 * cos (2 * z);  
0012  
0013 % Make window struct  
0014 w.type      = 'SFT3M';  
0015 w.alpha     = 0;  
0016 w.psll      = 44.2;  
0017 w.rov       = 65.5;  
0018 w.nenbw     = 2.9452;  
0019 w.w3db      = 2.9183;  
0020 w.flatness  = -0.0115;  
0021 w.ws        = sum(v);  
0022 w.ws2       = sum(v.*v);  
0023 w.win       = v;  
0024  
0025 % END
```


win_SFT4F

PURPOSE ^

WIN_SFT4F return SFT4F window sample for input z.

SYNOPSIS ^

function w = win_SFT4F(w, z)

DESCRIPTION ^

```
WIN_SFT4F return SFT4F window sample for input z.  
M Hewitson 18-05-07  
$Id: win_SFT4F.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_SFT4F(w, z)  
0002  
0003 % WIN_SFT4F return SFT4F window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_SFT4F.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $  
0008 %  
0009  
0010 z = z * 2 * pi;  
0011 v = 0.21706 - 0.42103 * cos (z) + 0.28294 * cos (2 * z) - 0.07897 * cos (3 * z);  
0012  
0013 % Make window struct  
0014 w.type      = 'SFT4F';  
0015 w.alpha    = 0;  
0016 w.psll     = 44.7;  
0017 w.rov      = 75;  
0018 w.nenbw    = 3.7970;  
0019 w.w3db     = 3.7618;  
0020 w.flatness = 0.0041;  
0021 w.ws       = sum(v);  
0022 w.ws2      = sum(v.*v);  
0023 w.win       = v;  
0024  
0025 % END
```


win_SFT4M

PURPOSE ^

WIN_SFT4M return SFT4M window sample for input z.

SYNOPSIS ^

function w = win_SFT4M(w, z)

DESCRIPTION ^

```
WIN_SFT4M return SFT4M window sample for input z.  
M Hewitson 18-05-07  
$Id: win_SFT4M.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_SFT4M(w, z)  
0002  
0003 % WIN_SFT4M return SFT4M window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_SFT4M.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $  
0008 %  
0009 %  
0010 z = z * 2 * pi;  
0011 v = 0.241906 - 0.460841 * cos(z) ...  
0012 + 0.255381 * cos (2 * z) - 0.041872 * cos (3 * z);  
0013  
0014 % Make window struct  
0015 w.type      = 'SFT4M';  
0016 w.alpha     = 0;  
0017 w.psll      = 66.5;  
0018 w.rov       = 72.1;  
0019 w.nenbw     = 3.3868;  
0020 w.w3db      = 3.3451;  
0021 w.flatness  = -0.0067;  
0022 w.ws       = sum(v);  
0023 w.ws2       = sum(v.*v);  
0024 w.win       = v;  
0025  
0026 % END
```


win_SFT5F

PURPOSE ^

WIN_SFT5F return SFT5F window sample for input z.

SYNOPSIS ^

function w = win_SFT5F(w, z)

DESCRIPTION ^

```
WIN_SFT5F return SFT5F window sample for input z.  
M Hewitson 18-05-07  
$Id: win_SFT5F.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_SFT5F(w, z)  
0002  
0003 % WIN_SFT5F return SFT5F window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_SFT5F.m,v 1.1 2008/06/20 10:46:55 hewitson Exp $  
0008 %  
0009 %  
0010 z = z * 2 * pi;  
0011 v = 0.1881 - 0.36923 * cos (z) + ...  
0012      0.28702 * cos (2 * z) - 0.13077 * cos (3 * z) + 0.02488 * cos (4 * z);  
0013  
0014 % Make window struct  
0015 w.type      = 'SFT5F';  
0016 w.alpha     = 0;  
0017 w.psll      = 57.3;  
0018 w.rov       = 78.5;  
0019 w.nenbw     = 4.3412;  
0020 w.w3db      = 4.2910;  
0021 w.flatness  = -0.0025;  
0022 w.ws        = sum(v);  
0023 w.ws2       = sum(v.*v);  
0024 w.win       = v;  
0025  
0026 % END
```


win_SFT5M

PURPOSE [^]

WIN_SFT5M return SFT5M window sample for input z.

SYNOPSIS [^]

function w = win_SFT5M(w, z)

DESCRIPTION [^]

```
WIN_SFT5M return SFT5M window sample for input z.  
M Hewitson 18-05-07  
$Id: win_SFT5M.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE [^]

```
0001 function w = win_SFT5M(w, z)  
0002  
0003 % WIN_SFT5M return SFT5M window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_SFT5M.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $  
0008 %  
0009  
0010 z = z * 2 * pi;  
0011 v = 0.209671 - 0.407331 * cos(z) + ...  
0012      0.281225 * cos (2 * z) - 0.092669 * cos (3 * z) + ...  
0013      0.0091036 * cos (4 * z);  
0014  
0015 % Make window struct  
0016 w.type      = 'SFT5M';  
0017 w.alpha     = 0;  
0018 w.psll      = 89.9;  
0019 w.rov       = 76;  
0020 w.nenbw     = 3.8852;  
0021 w.w3db     = 3.8340;  
0022 w.flatness = 0.0039;  
0023 w.ws       = sum(v);  
0024 w.ws2      = sum(v.*v);  
0025 w.win      = v;  
0026  
0027 % END
```


win_Welch

PURPOSE ^

WIN_WELCH return Welch window sample for input z.

SYNOPSIS ^

function w = win_Welch(w, z)

DESCRIPTION ^

```
WIN_WELCH return Welch window sample for input z.  
M Hewitson 18-05-07  
$Id: win_Welch.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin](#) % SPECWIN spectral window object class constructor.

SOURCE CODE ^

```
0001 function w = win_Welch(w, z)  
0002  
0003 % WIN_WELCH return Welch window sample for input z.  
0004 %  
0005 % M Hewitson 18-05-07  
0006 %  
0007 % $Id: win_Welch.m,v 1.1 2008/06/20 10:46:54 hewitson Exp $  
0008 %  
0009  
0010 v = 1 - (2 * z - 1).^2;  
0011  
0012 % Make window struct  
0013 w.type      = 'Welch';  
0014 w.alpha     = 0;  
0015 w.psll      = 21.3;  
0016 w.rov       = 29.3;  
0017 w.nenbw     = 1.2;  
0018 w.w3db      = 1.1535;  
0019 w.flatness  = -2.2248;  
0020 w.ws        = sum(v);  
0021 w.ws2       = sum(v.*v);  
0022 w.win       = v;  
0023  
0024 % END
```


ltpda_obj

PURPOSE ^

LTPDA_OBJ is the abstract ltpda base class.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_OBJ is the abstract ltpda base class.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION:  LTPDA_OBJ is the ltpda base class.
               This ltpda_obj class is an abstract class and it is
               not possible to create an instance of this class.

SUPER CLASSES: handle (MATLAB class)

SUB CLASSES:  ltpda_nuc, ltpda_uc

PROPERTIES:

    Protected Properties (read only)
        version          - cvs-version string.

LTPDA_OBJ METHODS:

    Public Methods:
        eq                - equal operator for ltpda objects.
        isprop            - tests if the given field is one of the object properties.
        ne                - not equal operator for ltpda objects.

    Protected Methods:
        setVersion        - set the property 'version'
        prependVersion    - prepend the version string

    Abstract Methods:
        char               - returns one character string which represents the object
        copy               - copies an object
        display            - displays an object
        update_struct      - updates a object structure to the current tbx-version

REMARK:       It is necessary to define the ABSTRACT methods and properties in the
               sub-classes because if they are not defined is the sub-class as well
               an abstract class. (See ltpda_nuo)

M-FILE INFO:  The following call returns an minfo object that contains
               information about the ltpda_obj constructor:
               >> info = ltpda_obj.getInfo
               or  >> info = ltpda_obj.getInfo('ltpda_obj')

VERSION:      $Id: ltpda_obj.m,v 1.22 2008/09/06 12:24:27 hewitson Exp $

HISTORY:      19-05-2008 Diepholz
               Creation.

SEE ALSO:     ltpda_nuo, ltpda_uo, handle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [eq](#) EQ overloads the == operator for ltpda objects.
-  [isprop](#) ISPROP tests if the given field is one of the object properties.

-  `ltpda_obj` LTPDA_OBJ is the abstract ltpda base class.
-  `ne` NE overloads the ~= operator for ltpda objects.
-  `prependVersion` PREPENDVERSION Prepend the input-version to the property 'version'
-  `setVersion` SETVERSION Set the property 'version'

This function is called by:

-  `ltpda_obj` LTPDA_OBJ is the abstract ltpda base class.

SUBFUNCTIONS ^

-  `function obj = ltpda_obj(varargin)`
-  `function ii = getInfo(varargin)`
-  `function out = VEROUT()`
-  `function out = SETS()`
-  `function out = getDefaultPlist()`

SOURCE CODE ^

```
0001 % LTPDA_OBJ is the abstract ltpda base class.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION:    LTPDA_OBJ is the ltpda base class.
0005 %                  This ltpda_obj class is an abstract class and it is
0006 %                  not possible to create an instance of this class.
0007 %
0008 % SUPER CLASSES: handle (MATLAB class)
0009 %
0010 % SUB CLASSES:   ltpda_nuc, ltpda_uc
0011 %
0012 % PROPERTIES:
0013 %
0014 %     Protected Properties (read only)
0015 %         version          - cvs-version string.
0016 %
0017 % LTPDA_OBJ METHODS:
0018 %
0019 %     Public Methods:
0020 %         eq                - equal operator for ltpda objects.
0021 %         isprop            - tests if the given field is one of the object properties.
0022 %         ne                - not equal operator for ltpda objects.
0023 %
0024 %     Protected Methods:
0025 %         setVersion        - set the property 'version'
0026 %         prependVersion    - prepend the version string
0027 %
0028 %     Abstract Methods:
0029 %         char              - returns one character string which represents the object
0030 %         copy              - copies an object
0031 %         display           - displays an object
0032 %         update_struct     - updates a object structure to the current tbx-version
0033 %
0034 % REMARK:           It is necessary to define the ABSTRACT methods and properties in the
0035 %                   sub-classes because if they are not defined is the sub-class as well
0036 %                   an abstract class. (See ltpda_nuo)
0037 %
0038 % M-FILE INFO:      The following call returns an minfo object that contains
0039 %                   information about the ltpda_obj constructor:
0040 %                   >> info = ltpda_obj.getInfo
0041 %                   or   >> info = ltpda_obj.getInfo('ltpda_obj')
0042 %
0043 % VERSION:          $Id: ltpda_obj.m,v 1.22 2008/09/06 12:24:27 hewitson Exp $
0044 %
0045 % HISTORY:          19-05-2008 Diepholz
0046 %                   Creation.
0047 %
0048 % SEE ALSO:         ltpda_nuo, ltpda_uo, handle
0049 %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051 %
0052 classdef ltpda_obj < handle
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055 %                                     Property definition %
```

```
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 %----- Public (read/write) Properties -----
0059 properties
0060 end
0061
0062 %----- Protected read-only Properties -----
0063 properties (SetAccess = protected)
0064 end
0065
0066 %----- Abstract Properties -----
0067 properties (Abstract = true, SetAccess = protected)
0068 % version
0069 end
0070
0071 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0072 % Check property setting %
0073 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0074
0075 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076 % Constructor %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
0079 methods
0080 function obj = ltpda_obj(varargin)
0081
0082     %%%%%%%%%%% Set default values %%%%%%%%%%%
0083
0084     if nargin == 1
0085
0086         if isstruct(varargin{1})
0087             %%%%%%%%%%% obj = ltpda_uo(struct) %%%%%%%%%%%
0088             end
0089
0090         end
0091
0092     end
0093 end
0094
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096 % Methods (public) %
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098
0099 methods (Access = public)
0100     result = eq(obj1, obj2, varargin)
0101     result = ne(obj1, obj2, varargin)
0102     varargout = isprop(varargin)
0103 end
0104
0105 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0106 % Methods (protected) %
0107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0108
0109 methods (Access = protected)
0110     obj = prependVersion(obj, val)
0111     setVersion(obj, val)
0112 end
0113
0114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0115 % Methods (static) %
0116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0117
0118 methods (Static)
0119
0120     function ii = getInfo(varargin)
0121         ii = utils.helper.generic_getInfo(varargin{:}, 'ltpda_obj');
0122     end
0123
0124     function out = VEROUT()
0125         out = '$Id: ltpda_obj.m,v 1.22 2008/09/06 12:24:27 hewitson Exp $';
0126     end
0127
0128     function out = SETS()
0129         out = {};
0130     end
0131
0132     function out = getDefaultPlist()
0133         out = [];
0134     end
0135
0136 end
0137
0138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0139 % Methods (abstract) %
0140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141
```



```
0142     methods (Abstract)
0143         varargout = char(varargin)
0144         varargout = copy(varargin)
0145         txt       = display(varargin)
0146     end
0147
0148     methods (Abstract = true, Static = true)
0149         varargout = update_struct(varargin);
0150     end
0151
0152 end
0153
```

ltpda_uoh

PURPOSE ^

LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_UOH is the ltpda base class for ltpda user object classes
              This class is an abstract class and it is not possible to create
              an instance of this class.

SUPER CLASSES: ltpda_uo < ltpda_obj

SUB CLASSES:   ao, miir, mfir, specwin, timespan, pzmodel, history, ssm,

PROPERTIES:

  Inherit Properties (read only)
    name      - name of object
    created   - creation time (time-object)
    prov      - contains a instance of the provenance class.
    hist      - history of the object (history object)
    version   - cvs-version string.

  Protected Properties (read only)
    hist      - history of the object (history object)

LTPDA_UOH Methods:

  Protected Methods
    setHist    - set the property 'hist'
    addHistory - add a history to a ltpda_uo object

  Public Methods
    setName    - set the property 'name' - Overloading the method in ltpda_uo
                  This set-functions adds the history

M-FILE INFO: The following call returns an minfo object that contains
              information about the ltpda_uoh constructor:
              >> info = ltpda_uoh.getInfo
              or   >> info = ltpda_uoh.getInfo('ltpda_uoh')

VERSION:      $Id: ltpda_uoh.m,v 1.14 2008/09/03 16:34:07 hewitson Exp $

HISTORY:      19-05-2008 Diepholz
               Creation.

SEE ALSO:     ltpda_obj, ao, miir, mfir, specwin, timespan, pzmodel, history, ssm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [addHistory](#) ADDHISTORY Add a history-object to the ltpda_uo object.
-  [fromDataInMAT](#) FROMDATAINMAT Default method to convert a data-array into am ltpda_uoh
-  [fromDatafile](#) FROMDATAFILE Default method to convert a data-file into a ltpda_uoh-object
-  [fromFile](#) Construct a ltpda_ob from a file
-  [fromLISO](#) FROMLISO Default method to read LISO files

-  [fromRepository](#) Retrieve a ltpda_uo from a repository
-  [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
-  [setHist](#) SETHIST Set the property 'hist'
-  [setName](#) SETNAME Set the property 'name'.

This function is called by:

-  [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SUBFUNCTIONS ^

-  [function obj = set.hist\(obj, val\)](#)
-  [function obj = ltpda_uoh\(varargin\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LTPDA_UOH is the ltpda base class for ltpda user object classes
0005 %               This class is an abstract class and it is not possible to create
0006 %               an instance of this class.
0007 %
0008 % SUPER CLASSES: ltpda_uo < ltpda_obj
0009 %
0010 % SUB CLASSES:   ao, miir, mfir, specwin, timespan, pzmodel, history, ssm,
0011 %
0012 % PROPERTIES:
0013 %
0014 %   Inherit Properties (read only)
0015 %       name      - name of object
0016 %       created   - creation time (time-object)
0017 %       prov      - contains a instance of the provenance class.
0018 %       hist      - history of the object (history object)
0019 %       version   - cvs-version string.
0020 %
0021 %   Protected Properties (read only)
0022 %       hist      - history of the object (history object)
0023 %
0024 % LTPDA_UOH Methods:
0025 %
0026 %   Protected Methods
0027 %       setHist    - set the property 'hist'
0028 %       addHistory - add a history to a ltpda_uo object
0029 %
0030 %   Public Methods
0031 %       setName    - set the property 'name' - Overloading the method in ltpda_uo
0032 %                   This set-functions adds the history
0033 %
0034 % M-FILE INFO: The following call returns an minfo object that contains
0035 %               information about the ltpda_uoh constructor:
0036 %               >> info = ltpda_uoh.getInfo
0037 %               or   >> info = ltpda_uoh.getInfo('ltpda_uoh')
0038 %
0039 % VERSION:     $Id: ltpda_uoh.m,v 1.14 2008/09/03 16:34:07 hewitson Exp $
0040 %
0041 % HISTORY:     19-05-2008 Diepholz
0042 %               Creation.
0043 %
0044 % SEE ALSO:    ltpda_obj, ao, miir, mfir, specwin, timespan, pzmodel, history, ssm
0045 %
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %
0048 classdef ltpda_uoh < ltpda_uo
0049 %
0050 %
0051 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052 %               Property definition
0053 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0054
0055 %----- Public (read/write) Properties -----
0056 properties
0057 end
0058
0059 %----- Protected read-only Properties -----
0060 properties (SetAccess = protected)
0061     hist = '';
0062 end
0063
0064 %----- Private Properties -----
0065 properties (GetAccess = protected, SetAccess = protected)
0066 end
0067
0068 %----- Abstract Properties -----
0069 properties (Abstract = true, SetAccess = protected)
0070 end
0071
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073 %                               Check property setting                               %
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075
0076 methods
0077     function obj = set.hist(obj, val)
0078         if ~(isa(val, 'history') || isempty(val))
0079             error('### The value for the property ''hist'' must be\n### a history-object or
empty but it is\n### from the class %s', class(val));
0080         end
0081         obj.hist = val;
0082     end
0083 end
0084
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086 %                               Constructor                               %
0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088
0089 methods
0090     function obj = ltpda_uoh(varargin)
0091
0092         %% Call superclass
0093         obj = obj@ltpda_uo(varargin{:});
0094
0095         %%%%%%%%% Set default values %%%%%%%%%
0096
0097         if nargin == 1
0098
0099             if isstruct(varargin{1})
0100                 %%%%%%%%% obj = ltpda_uoh(struct) %%%%%%%%%
0101                 %% Set properties which are declared in this class
0102                 obj.hist = utils.helper.struct2obj(varargin{1}.hist, 'history');
0103             end
0104
0105         end
0106
0107     end
0108 end
0109
0110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111 %                               Methods (public)                               %
0112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0113
0114 methods
0115 end
0116
0117 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0118 %                               Methods (protected)                               %
0119 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0120
0121 methods (Access = protected)
0122     varargout = setHist(obj, val)
0123     obj = fromFile(pbj, pli)
0124     obj = fromRepository(obj, pli)
0125     obj = fromLISO(obj, filename)
0126     obj = fromDataInMAT(obj, data, filename)
0127     obj = fromDatafile(obj, pli)
0128 end
0129
0130 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0131 %                               Methods (public)                               %
0132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0133
0134 methods
0135     varargout = addHistory(varargin)
0136     varargout = setName(varargin)
0137 end
0138

```

```
0139
0140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141 %                                     Methods (static)                                     %
0142 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0143
0144 methods (Static)
0145     function ii = getInfo(varargin)
0146         ii = utils.helper.generic_getInfo(varargin{:}, 'ltpda_uoh');
0147     end
0148
0149     function out = VEROUT()
0150         out = '$Id: ltpda_uoh.m,v 1.14 2008/09/03 16:34:07 hewitson Exp $';
0151     end
0152
0153     function out = SETS()
0154         out = {};
0155     end
0156
0157     function out = getDefaultPlist()
0158         out = [];
0159     end
0160
0161 end
0162
0163 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0164 %                                     Methods (abstract)                                     %
0165 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0166
0167 methods (Abstract)
0168 end
0169
0170 end
0171
```

ltpda_uo

PURPOSE ^

LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_UO is the abstract ltpda base class for ltpda user object classes.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_UO is the ltpda base class for ltpda user object classes.
              This class is an abstract class and it is not possible to create
              an instance of this class.
              ALL ltpda user object classes inherit this 'abstract' class.

SUPER CLASSES: ltpda_obj

SUB CLASSES:   ltpda_uoh, plist

PROPERTIES:

    Inherit Properties (read only)
        version - cvs-version string.

    Protected Properties (read only)
        name     - name of object
        created  - creation time (time-object)
        creator  - contains a instance of the provenance class.

LTPDA_UO Methods:

    Public Methods
        setName  - set the property 'name'
        save     - save ltpda user objects
        submit   - submits the given collection of objects to an LTPDA Repository.

    Protected Methods
        setCreated - set the property 'created'
        setCreator - set the property 'creator'

    Static Methods
        retrieve   - retrieve ltpda objects with given ids from the repository

    Abstract Methods
        string     - writes a command string that can be used to recreate the object

M-FILE INFO: The following call returns an minfo object that contains
              information about the ltpda_uo constructor:
                >> info = ltpda_uo.getInfo
              or  >> info = ltpda_uo.getInfo('ltpda_uo')

VERSION:      $Id: ltpda_uo.m,v 1.23 2008/09/03 16:33:56 hewitson Exp $

HISTORY:      19-05-2008 Diepholz
               Creation.

SEE ALSO: ltpda_obj, ltpda_uoh, plist
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

-  **retrieve** RETRIEVE retrieves a collection of objects from an LTPDA repository.
-  **save** SAVE overloads save operator for ltpda objects.
-  **setCreated** SETCREATED Set the property 'created'
-  **setCreator** SETCREATOR Set the property 'creator'
-  **setName** SETNAME Set the property 'name'.
-  **submit** SUBMIT submits the given collection of objects to an LTPDA Repository.

This function is called by:

-  **ltpda_uo** LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

SUBFUNCTIONS ^

-  **function** obj = ltpda_uo(varargin)
-  **function** ii = getInfo(varargin)
-  **function** out = VEROUT()
-  **function** out = SETS()
-  **function** out = getDefaultPlist()

SOURCE CODE ^

```
0001 % LTPDA_UO is the abstract ltpda base class for ltpda user object classes.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LTPDA_UO is the ltpda base class for ltpda user object classes.
0005 %               This class is an abstract class and it is not possible to create
0006 %               an instance of this class.
0007 %               ALL ltpda user object classes inherit this 'abstract' class.
0008 %
0009 % SUPER CLASSES: ltpda_obj
0010 %
0011 % SUB CLASSES:   ltpda_uoh, plist
0012 %
0013 % PROPERTIES:
0014 %
0015 %     Inherit Properties (read only)
0016 %         version - cvs-version string.
0017 %
0018 %     Protected Properties (read only)
0019 %         name     - name of object
0020 %         created  - creation time (time-object)
0021 %         creator  - contains a instance of the provenance class.
0022 %
0023 % LTPDA_UO Methods:
0024 %
0025 %     Public Methods
0026 %         setName  - set the property 'name'
0027 %         save     - save ltpda user objects
0028 %         submit   - submits the given collection of objects to an LTPDA Repository.
0029 %
0030 %     Protected Methods
0031 %         setCreated - set the property 'created'
0032 %         setCreator - set the property 'creator'
0033 %
0034 %     Static Methods
0035 %         retrieve   - retrieve ltpda objects with given ids from the repository
0036 %
0037 %     Abstract Methods
0038 %         string     - writes a command string that can be used to recreate the object
0039 %
0040 % M-FILE INFO: The following call returns an minfo object that contains
0041 %               information about the ltpda_uo constructor:
0042 %               >> info = ltpda_uo.getInfo
0043 %               or   >> info = ltpda_uo.getInfo('ltpda_uo')
0044 %
0045 % VERSION:      $Id: ltpda_uo.m,v 1.23 2008/09/03 16:33:56 hewitson Exp $
0046 %
0047 % HISTORY:      19-05-2008 Diepholz
0048 %               Creation.
0049 %
0050 % SEE ALSO: ltpda_obj, ltpda_uoh, plist
0051 %
```

```

0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053
0054 classdef ltpda_uo < ltpda_obj
0055
0056
0057 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058 %                                     Property definition                                     %
0059 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 %----- Public (read/write) Properties -----
0062 properties
0063 end
0064
0065 %----- Protected read-only Properties -----
0066 properties (SetAccess = protected)
0067     name = 'none';
0068     created = time(0);
0069     creator = provenance();
0070 end
0071
0072 %----- Private Properties -----
0073 properties (GetAccess = protected, SetAccess = protected)
0074 end
0075
0076 %----- Abstract Properties -----
0077 properties (Abstract = true, SetAccess = protected)
0078 end
0079
0080 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0081 %                                     Check property setting                                     %
0082 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0083
0084 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0085 %                                     Constructor                                     %
0086 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0087
0088 methods
0089     function obj = ltpda_uo(varargin)
0090
0091         %%% Call superclass
0092         obj = obj@ltpda_obj(varargin{:});
0093
0094         %%% Set default values %%%
0095         %%% Exept for a struct as an input
0096         if ~(nargin == 1 && isstruct(varargin{1}))
0097             % Set created time
0098             obj.created = time();
0099         end
0100
0101         if nargin == 1
0102
0103             if isstruct(varargin{1})
0104                 %%% obj = ltpda_uo(struct) %%%
0105
0106                 %%% Set properties which are declared in this class
0107                 uo_struct = varargin{1};
0108
0109                 obj.name = uo_struct.name;
0110                 obj.created = utils.helper.struct2obj(uo_struct.created, 'time');
0111                 obj.creator = utils.helper.struct2obj(uo_struct.creator, 'provenance');
0112             end
0113         end
0114     end
0115 end
0116
0117 end
0118
0119 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0120 %                                     Methods (public)                                     %
0121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122
0123 methods
0124     varargout = submit(varargin)
0125     varargout = setName(varargin)
0126     varargout = save(varargin)
0127 end
0128
0129 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0130 %                                     Methods (protected)                                     %
0131 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0132
0133 methods (Access = protected)
0134     varargout = setCreated(obj, val)
0135     varargout = setCreator(obj, val)
0136 end
0137

```



```
0138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
0139 %                                Methods (private)                                %  
0140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
0141  
0142 methods (Access = protected)  
0143 end  
0144  
0145 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
0146 %                                Methods (static)                                %  
0147 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
0148  
0149 methods (Static)  
0150  
0151     varargout = retrieve(varargin)  
0152  
0153     function ii = getInfo(varargin)  
0154         ii = utils.helper.generic_getInfo(varargin{:}, 'ltpda_uo');  
0155     end  
0156  
0157     function out = VEROUT()  
0158         out = '$Id: ltpda_uo.m,v 1.23 2008/09/03 16:33:56 hewitson Exp $';  
0159     end  
0160  
0161     function out = SETS()  
0162         out = {};  
0163     end  
0164  
0165     function out = getDefaultPlist()  
0166         out = [];  
0167     end  
0168  
0169 end  
0170  
0171 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
0172 %                                Methods (abstract)                                %  
0173 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
0174  
0175 methods (Abstract)  
0176     cmd = string(obj, varargin)  
0177 end  
0178  
0179 end  
0180  
0181
```

cell_add

PURPOSE ^

adds corresponding matrices of same sizes or empty inside cell array

SYNOPSIS ^

function a = cell_add(a,b)

DESCRIPTION ^

```
adds corresponding matrices of same sizes or empty inside cell array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: cell_add adds corresponding matrices of same sizes or empty inside
cell array

CALL: [cell3] = ssm.cell_add(cell1,cell2)

INPUTS:
    cell1 - cell array of matrices representing a matrix by blocs.
           blocs may be empty
    cell2 - cell array of matrices representing a matrix by blocs.
           blocs may be empty

OUTPUTS:
    cell3 - cell array of matrices representing a matrix by blocs.
           blocs may be empty

***** There are no parameters *****

VERSION: '$Id: $'

HISTORY:
07-08-2008

TO DO :
check ME in case of mixed symbolic and double
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 % adds corresponding matrices of same sizes or empty inside cell array
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: cell_add adds corresponding matrices of same sizes or empty inside
0005 % cell array
0006 %
0007 % CALL: [cell3] = ssm.cell_add(cell1,cell2)
0008 %
0009 % INPUTS:
0010 %     cell1 - cell array of matrices representing a matrix by blocs.
0011 %           blocs may be empty
0012 %     cell2 - cell array of matrices representing a matrix by blocs.
0013 %           blocs may be empty
0014 %
0015 % OUTPUTS:
```

```
0016 %           cell3 - cell array of matrices representing a matrix by blocs.
0017 %                   blocs may be empty
0018 %
0019 % ***** There are no parameters *****
0020 %
0021 % VERSION: '$Id: $'
0022 %
0023 %
0024 % HISTORY:
0025 % 07-08-2008
0026 %
0027 % TO DO :
0028 % check ME in case of mixed symbolic and double
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031
0032 function a = cell_add(a,b)
0033 Nrow = size(a,1);
0034 Ncol = size(a,2);
0035 for i=1:Nrow
0036     for j=1:Ncol
0037         if isequal(a{i,j}, [])
0038             if ~isequal(b{i,j}, [])
0039                 a{i,j}=b{i,j};
0040             end
0041         else
0042             if ~isequal(b{i,j}, [])
0043                 try
0044                     a{i,j} = a{i,j} + b{i,j};
0045                 catch ME
0046                     a{i,j} = sym(a{i,j}) + sym(b{i,j})
0047                 end
0048             end
0049         end
0050     end
0051 end
0052 end
```

cell_fusion

PURPOSE ^

fusions a block defined matrix stored inside cell array into one matrix

SYNOPSIS ^

function a_out = cell_fusion(a, rowsizes, colsizes)

DESCRIPTION ^

```
fusions a block defined matrix stored inside cell array into one matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: cell_fusion fusions a block defined matrix stored inside cell
array into one matrix

CALL: [mat3] = ssm.cell_fusion(cell1, rowsizes, colsizes)

INPUTS:
    cell1      - cell array of matrices representing a matrix by blocs.
                 blocs may be empty
    rowsizes   - vector giving block height
    colsizes   - vector giving block width

OUTPUTS:
    mat3 - double or symbolic array

***** There are no parameters *****

VERSION: '$Id: $'

HISTORY:
07-08-2008

TO DO :
check ME in case of mixed symbolic and double
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
to deal with matrices whose size is not defined
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 % fusions a block defined matrix stored inside cell array into one matrix
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: cell_fusion fusions a block defined matrix stored inside cell
0005 % array into one matrix
0006 %
0007 % CALL: [mat3] = ssm.cell_fusion(cell1, rowsizes, colsizes)
0008 %
0009 % INPUTS:
0010 %     cell1      - cell array of matrices representing a matrix by blocs.
0011 %                 blocs may be empty
0012 %     rowsizes   - vector giving block height
0013 %     colsizes   - vector giving block width
0014 %
0015 % OUTPUTS:
```

```

0016 %      mat3 - double or symbolic array
0017 %
0018 % ***** There are no parameters *****
0019 %
0020 % VERSION: '$Id: $'
0021 %
0022 %
0023 % HISTORY:
0024 % 07-08-2008
0025 %
0026 % TO DO :
0027 % check ME in case of mixed symbolic and double
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 function a_out = cell_fusion(a, rowsizes, colsizes)
0030 % to deal with matrices whose size is not defined
0031 rowsizes = max(zeros(size(rowsizes)), rowsizes);
0032 colsizes = max(zeros(size(colsizes)), colsizes);
0033 rowrank = cumsum([1 rowsizes]);
0034 colrank = cumsum([1 colsizes]);
0035 Nrow = length(rowsizes);
0036 Ncol = length(colsizes);
0037 a_out = zeros(rowrank(Nrow+1)-1, colrank(Ncol+1)-1);
0038 for i=1:Nrow
0039     for j=1:Ncol
0040         if ~isempty(a{i,j})
0041             rowmin = rowrank(i);
0042             rowmax = rowrank(i+1)-1;
0043             colmin = colrank(j);
0044             colmax = colrank(j+1)-1;
0045             if rowmax>=rowmin && colmax>=colmin
0046                 try
0047                     a_out(rowmin:rowmax, colmin:colmax) = a{i,j};
0048                 catch ME
0049                     a_out = sym(a_out);
0050                     a_out(rowmin:rowmax, colmin:colmax) = sym(a{i,j});
0051             end
0052         end
0053     end
0054 end
0055 end
0056 end

```

cell_mult

PURPOSE ^

multiplies block defined matrix stored inside cell array

SYNOPSIS ^

function c = cell_mult(a,b)

DESCRIPTION ^

```
multiplies block defined matrix stored inside cell array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: cell_mult multiplies block defined matrix stored inside cell
array

CALL: [cell3] = ssm.cell_mult(cell1,cell2)

INPUTS:
    cell1 - cell array of matrices representing a matrix by blocs.
           blocs may be empty
    cell2 - cell array of matrices representing a matrix by blocs.
           blocs may be empty

OUTPUTS:
    cell3 - cell array of matrices representing a matrix by blocs.
           blocs may be empty

***** There are no parameters *****

VERSION: '$Id: $'

HISTORY:
07-08-2008

TO DO :
check ME in case of mixed symbolic and double
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 % multiplies block defined matrix stored inside cell array
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: cell_mult multiplies block defined matrix stored inside cell
0005 % array
0006 %
0007 % CALL: [cell3] = ssm.cell_mult(cell1,cell2)
0008 %
0009 % INPUTS:
0010 %     cell1 - cell array of matrices representing a matrix by blocs.
0011 %           blocs may be empty
0012 %     cell2 - cell array of matrices representing a matrix by blocs.
0013 %           blocs may be empty
0014 %
0015 % OUTPUTS:
```

```
0016 %      cell3 - cell array of matrices representing a matrix by blocs.
0017 %              blocs may be empty
0018 %
0019 % ***** There are no parameters *****
0020 %
0021 % VERSION: '$Id: $'
0022 %
0023 %
0024 % HISTORY:
0025 % 07-08-2008
0026 %
0027 % TO DO :
0028 % check ME in case of mixed symbolic and double
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030 function c = cell_mult(a,b)
0031     n1 = size(a,1);
0032     n2 = size(a,2);
0033     n3 = size(b,2);
0034     c = cell(n1,n3);
0035     for i=1:n1
0036         for j=1:n3
0037             for k=1:n2
0038                 if ~(isequal(a{i,k}, []) || isequal(b{k,j}, []))
0039
0040                     if isempty(c{i,j})
0041                         c{i,j} = a{i,k}*b{k,j};
0042                     else
0043                         try
0044                             c{i,j} = c{i,j} + a{i,k}*b{k,j};
0045                         catch
0046                             c{i,j} = sym(c{i,j}) + sym(a{i,k})*sym(b{k,j});
0047                         end
0048                     end
0049                 end
0050             end
0051         end
0052     end
0053 end
```

cell_recut

PURPOSE ^

cuts a matrix into blocks stored inside cell array

SYNOPSIS ^

function a_out = cell_recut(a, rowsizes, colsizes)

DESCRIPTION ^

```
cuts a matrix into blocks stored inside cell array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: cell_recut  cuts a matrix into blocks stored inside cell
array. Blocks with zeros are nullified, except in the diagonal (to
preserve information on matrix size)

CALL: [cell3] = ssm.cell_recut(mat1,rowsizes,colsizes)

INPUTS:
    mat1      - numeric array to be cut in pieces
    rowsizes  - vector giving block height
    colsizes  - vector giving block width

OUTPUTS:
    cell3 - cell array of matrices representing a matrix by blocs.
           blocs may be empty

***** There are no parameters *****

VERSION: '$Id: $'

HISTORY:
07-08-2008

TO DO :
check ME in case of mixed symbolic and double
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
to deal with matrices whose size is not defined
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [double](#) Convert a statespace model object to double arrays for given i/o

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 % cuts a matrix into blocks stored inside cell array
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: cell_recut  cuts a matrix into blocks stored inside cell
0005 % array. Blocks with zeros are nullified, except in the diagonal (to
0006 % preserve information on matrix size)
0007 %
0008 % CALL: [cell3] = ssm.cell_recut(mat1,rowsizes,colsizes)
0009 %
0010 % INPUTS:
0011 %     mat1      - numeric array to be cut in pieces
```



```

0012 %      rowsizes - vector giving block height
0013 %      colsizes - vector giving block width
0014 %
0015 % OUTPUTS:
0016 %      cell3 - cell array of matrices representing a matrix by blocs.
0017 %      blocs may be empty
0018 %
0019 % ***** There are no parameters *****
0020 %
0021 % VERSION: '$Id: $'
0022 %
0023 %
0024 % HISTORY:
0025 % 07-08-2008
0026 %
0027 % TO DO :
0028 % check ME in case of mixed symbolic and double
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030 function a_out = cell_recut(a, rowsizes, colsizes)
0031 % to deal with matrices whose size is not defined
0032 rowsizes = max(zeros(size(rowsizes)), rowsizes);
0033 colsizes = max(zeros(size(colsizes)), colsizes);
0034 rowrank = cumsum([1 rowsizes]);
0035 colrank = cumsum([1 colsizes]);
0036 Nrow = length(rowsizes);
0037 Ncol = length(colsizes);
0038 a_out = cell(Nrow, Ncol);
0039 for i=1:Nrow
0040     for j=1:Ncol
0041         rowmin = rowrank(i);
0042         rowmax = rowrank(i+1)-1;
0043         colmin = colrank(j);
0044         colmax = colrank(j+1)-1;
0045
0046         if rowmax >= rowmin && colmax >= colmin
0047             if isa(a, 'sym')
0048                 try
0049                     a_out{i,j} = double(a(rowmin:rowmax, colmin:colmax));
0050                     if ~(i==j) && norm(a_out{i,j})==0
0051                         a_out{i,j}=[];
0052                     end
0053                 catch
0054                     a_out{i,j} = a(rowmin:rowmax, colmin:colmax);
0055                 end
0056             elseif ~norm(a(rowmin:rowmax, colmin:colmax))==0
0057                 a_out{i,j} = double(a(rowmin:rowmax, colmin:colmax));
0058             elseif i==j
0059                 a_out{i,j} = zeros(rowsizes(i), colsizes(j));
0060             end
0061             elseif i~=j
0062                 a_out{i,j} = zeros(rowsizes(i), colsizes(j));
0063             end
0064         end
0065     end
0066 end

```

cell_select

PURPOSE ^

CELL_SELECT selects lines and columns of a block defined matrix stored in

SYNOPSIS ^

function [cell_mat_out, rowsizes_out, colsizes_out] = cell_select(varargin)

DESCRIPTION ^

```
CELL_SELECT selects lines and columns of a block defined matrix stored in
a cell array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: cell_select selects lines and columns of a blk defined
matrix stored in a cell array

CALL: [cell2] = ssm.cell_recut(cell1,rowsizes,colsizes)

INPUTS:
    cell1      - block defined matrix in cell array
    rowsizes   - vector giving block height
    colsizes   - vector giving block width

OUTPUTS:
    cell2      - cell array of matrices representing a matrix by blocs.
                  blocs may be empty

***** There are no parameters *****

VERSION: '$Id: $'

HISTORY:
12-08-2008

TO DO :
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 % CELL_SELECT selects lines and columns of a block defined matrix stored in
0002 % a cell array
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: cell_select selects lines and columns of a blk defined
0006 % matrix stored in a cell array
0007 %
0008 % CALL: [cell2] = ssm.cell_recut(cell1,rowsizes,colsizes)
0009 %
0010 % INPUTS:
0011 %     cell1      - block defined matrix in cell array
0012 %     rowsizes   - vector giving block height
0013 %     colsizes   - vector giving block width
0014 %
0015 % OUTPUTS:
0016 %     cell2      - cell array of matrices representing a matrix by blocs.
```

```

0017 %                blocs may be empty
0018 %
0019 % ***** There are no parameters *****
0020 %
0021 % VERSION: '$Id: $'
0022 %
0023 %
0024 % HISTORY:
0025 % 12-08-2008
0026 %
0027 % TO DO :
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 function [cell_mat_out, rowsizes_out, colsizes_out] = cell_select(varargin)
0030
0031     cell_mat_in = varargin{1};
0032     rowsizes = varargin{2};
0033     colsizes = varargin{3};
0034     rowselect = varargin{4};
0035     colselect = varargin{5};
0036
0037     %% field for rows
0038     Nrows = size(cell_mat_in, 1);
0039     rows_log = cell(1,Nrows);
0040     if strcmp(rowselect, 'ALL')
0041         for i=1:Nrows
0042             rows_log{i} = true(1,rowsizes(i));
0043         end
0044     elseif iscellstr(rowselect)
0045         rownames = varargin{6};
0046         for i=1:Nrows
0047             rows_log{i} = true(1, rowsizes(i));
0048             for j=1:rowsizes(i)
0049                 rows_log{i}(j) = (sum(ismember(rowselect, rownames{i}{j} ))>0);
0050             end
0051         end
0052     elseif isa(rowselect, 'cell')
0053         for i=1:Nrows
0054             if isa(rowselect{i},'double')
0055                 rows_log{i} = rowselect{i};
0056             elseif isa(rowselect{i},'logical')
0057                 rows_log{i} = rowselect{i};
0058             if ~length(rowselect{i})==rowsizes(i)
0059                 error(['parameter 'rowselect' is of wrong size for block numer ',
0060 num2str(i)]);
0061             end
0062             else
0063                 error('parameter 'rowselect' is of wrong type')
0064             end
0065         end
0066     else
0067         error('parameter 'rowselect' is of wrong type')
0068     end
0069
0070     %% field for cols
0071     Ncols = size(cell_mat_in, 2);
0072     cols_log = cell(1,Ncols);
0073     if strcmp(colselect, 'ALL')
0074         for i=1:Ncols
0075             cols_log{i} = true(1,colsizes(i));
0076         end
0077     elseif iscellstr(colselect)
0078         colnames = varargin{7};
0079         for i=1:Ncols
0080             cols_log{i} = true(1, colsizes(i));
0081             for j=1:colsizes(i)
0082                 cols_log{i}(j) = (sum(ismember(colselect, colnames{i}{j} ))>0);
0083             end
0084         end
0085     elseif isa(colselect, 'cell')
0086         for i=1:Ncols
0087             if isa(colselect{i},'double')
0088                 cols_log{i} = colselect{i};
0089             elseif isa(colselect{i},'logical')
0090                 cols_log{i} = colselect{i};
0091             if ~length(colselect{i})==colsizes(i)
0092                 error(['parameter 'colselect' is of wrong size for block numer ',
0093 num2str(i)]);
0094             end
0095             else
0096                 error('parameter 'colselect' is of wrong type')
0097             end
0098         end
0099     else
0100         error('parameter 'colselect' is of wrong type')
0101     end

```

```

0101 %% assigning content
0102 cell_mat_out = cell(Nrows, Ncols);
0103 for i=1:Nrows
0104     for j=1:Ncols
0105         if ~isequal(cell_mat_in{i,j}, [])
0106             if ~isempty(rows_log{i}) && ~isempty(cols_log{j})
0107                 cell_mat_out{i,j} = cell_mat_in{i,j}(rows_log{i}, cols_log{j});
0108             else
0109                 switch class(cell_mat_in{i,j})
0110                     case {'double', 'sym'}
0111                         cell_mat_out{i,j} = zeros( sum(rows_log{i}>0), sum(cols_log{j}>0));
0112                     case 'cell'
0113                         cell_mat_out{i,j} = cell( sum(rows_log{i}>0), sum(cols_log{j}>0));
0114                     end
0115                 end
0116             end
0117         end
0118     end
0119 end
0120 rowsizes_out = zeros(1, Nrows);
0121 for i=1:Nrows
0122     if isa(rows_log{i}, 'double')
0123         rowsizes_out(i) = length(rows_log{i});
0124     else
0125         rowsizes_out(i) = sum(rows_log{i});
0126     end
0127 end
0128
0129 colsizes_out = zeros(1, Ncols);
0130 for i=1:Ncols
0131     if isa(cols_log{i}, 'double')
0132         colsizes_out(i) = length(cols_log{i});
0133     else
0134         colsizes_out(i) = sum(cols_log{i});
0135     end
0136 end
0137 end

```

cellstrfind

PURPOSE ^

cellstrfind finds matching strings inside a cellstr array

SYNOPSIS ^

function varargout = cellstrfind(c,s,option)

DESCRIPTION ^

```
cellstrfind finds matching strings inside a cellstr array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: cellstrfind finds matching strings inside a cellstr array

CALL: [position, sum, value] = ssm.cellstrfind(c,s,option)

INPUTS:
    c      - cell array of strings
    s      - string to look for
    option - defines the output
              'position' > positions where s was found in c
              'sum'      > number of matches
              'value'    > logical array telling if s{i}==c
              'all'      > gives all three outputs above

OUTPUTS:
    position
    sum
    value
    as explained above, number of output depends on 'option field'

***** There are no parameters *****

VERSION: '$Id: $'

HISTORY:
07-08-2008

TO DO :
check ME in case of mixed symbolic and double
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 % cellstrfind finds matching strings inside a cellstr array
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: cellstrfind finds matching strings inside a cellstr array
0005 %
0006 % CALL: [position, sum, value] = ssm.cellstrfind(c,s,option)
0007 %
0008 % INPUTS:
0009 %      c      - cell array of strings
0010 %      s      - string to look for
0011 %      option - defines the output
```

```
0012 %           'position' > positions where s was found in c
0013 %           'sum'      > number of matches
0014 %           'value'    > logical array telling if s{i}==c
0015 %           'all'      > gives all three outputs above
0016 %
0017 % OUTPUTS:
0018 %     position
0019 %     sum
0020 %     value
0021 %     as explained above, number of output depends on 'option field'
0022 %
0023 % ***** There are no parameters *****
0024 %
0025 % VERSION: '$Id: $'
0026 %
0027 %
0028 % HISTORY:
0029 % 07-08-2008
0030 %
0031 % TO DO :
0032 % check ME in case of mixed symbolic and double
0033 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034 function varargout = cellstrfind(c,s,option)
0035     value = strcmp(c,s);
0036     position = find(value);
0037     sum = length(position);
0038     if strcmp(option, 'sum')
0039         varargout = {sum};
0040     elseif strcmp(option, 'value')
0041         varargout = {value};
0042     elseif strcmp(option, 'position')
0043         varargout = {position};
0044     elseif strcmp(option, 'all')
0045         varargout = {position, sum, value};
0046     end
0047 end
```

ssmFromBuiltinSystem

PURPOSE ^

%%%

SYNOPSIS ^

```
function varargout = ssmFromBuiltinSystem(pli)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      ssmFromBuiltinSystem
DESCRIPTION:   Construct a statespace model from one of the built-in
               systems.
CALL:         see ssm
PARAMETER:
VERSION:      $Id: ssmFromBuiltinSystem.m,v 1.41 2008/09/04 13:37:15 ingo Exp $
HISTORY:      15-04-2008 Hewitson
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ssm](#) SSM statespace model class constructor.

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      ssmFromBuiltinSystem
0004 %
```

```

0005 % DESCRIPTION: Construct a statespace model from one of the built-in
0006 % systems.
0007 %
0008 % CALL:      see ssm
0009 %
0010 % PARAMETER:
0011 %
0012 % VERSION:    $Id: ssmFromBuiltinSystem.m,v 1.41 2008/09/04 13:37:15 ingo Exp $
0013 %
0014 % HISTORY:    15-04-2008 Hewitson
0015 %             Creation
0016 %
0017 %%%%%%%%%%%
0018 function varargout = ssmFromBuiltinSystem(pli)
0019
0020     VERSION = '$Id: ssmFromBuiltinSystem.m,v 1.41 2008/09/04 13:37:15 ingo Exp $';
0021
0022     utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0023
0024     % get AO info
0025     ii = ssm.getInfo('ssm', 'From Built-in Model');
0026
0027     % Set the method version string in the minfo object
0028     ii.setMversion([VERSION '-->' ii.mversion]);
0029
0030     % Get default params
0031     pl = combine(pli, ii.plist);
0032
0033     % Get the name of the model the user wants
0034     bsys = find(pl, 'Built-in');
0035
0036     % Get a list of user model directories
0037     paths = getappdata(0, 'ltpda_ssm_model_paths');
0038     if ischar(paths)
0039         paths = {paths};
0040     end
0041
0042     % Always look in the system directory
0043     loc = mfilename('fullpath');
0044     idx = strfind(loc, filesep);
0045     loc = loc(1:idx(end));
0046     loc = fullfile(loc, '../..m/ssm_models');
0047     paths = [paths {loc}];
0048
0049     for jj=1:numel(paths)
0050         utils.helper.msg(utils.const.msg.PROC1, 'looking for models in %s', paths{jj});
0051     end
0052
0053     % list files in here
0054     models = {};
0055     for pp = 1:numel(paths)
0056         files = utils.prog.filescan(paths{pp}, '.m');
0057         for ff = 1:numel(files)
0058             [path, mname, ext, vers] = fileparts(files{ff});
0059             if strncmp(mname, 'ssm_model_', 10)
0060                 models = [models {strrep(mname, 'ssm_model_', '')}];
0061             end
0062         end
0063     end
0064
0065     % Give a list and an error if bsys is empty
0066     if isempty(bsys)
0067         for k=1:numel(models)
0068             disp(sprintf('%03d: %s', k, models{k}));
0069         end
0070         error('### No model specified. Please choose one from the list above.');
```



```
0071 end
0072
0073 % Find the matching model
0074 sys = [];
0075 if any(strcmp(models, bsys)) || (isnumeric(bsys) && bsys <= numel(models))
0076     if isnumeric(bsys)
0077         fcname = ['ssm_model_' models{bsys}];
0078     else
0079         fcname = ['ssm_model_' bsys];
0080     end
0081     [sys pl_dim_red] = feval(fcname, pl);
0082     sys.version = ii.mversion;
0083     sys.created = time(0);
0084     sys.creator = provenance();
0085     sys.hist = history(time().utc_epoch_milli, ii, pl, [], []);
0086
0087     % Convert to SSM object
0088     sys = ssm(sys);
0089     sys.reduce_model(pl_dim_red)
0090
0091 else
0092     for k=1:numel(models)
0093         disp(sprintf('%03d: %s', k, models{k}));
0094     end
0095     error('### Model not found. Please choose one from the above list.');
```

```
0096 end
0097
0098 % Set output
0099 varargout{1} = sys;
0100
0101 end
```

ssmFromDescription

PURPOSE ^

%%%

SYNOPSIS ^

```
function sys = ssmFromDescription(pli)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      ssmFromDescription
DESCRIPTION:   Construct a statespace model from a plist description
CALL:         see ssm
PARAMETER:
HISTORY:      15-04-2008 Adrien
              Creation
TODO:         inplement multiple i/o when subassign function is done
VERSION :     '$Id: ssmFromDescription.m,v 1.16 2008/08/30 19:45:08 adrien Exp $';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [char](#) CHAR convert a ssm object into a string.
-  [display](#) DISPLAY display ssm object.
-  [ssm](#) SSM statespace model class constructor.

This function is called by:

-  [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0002 %
0003 % FUNCTION:      ssmFromDescription
0004 %
0005 % DESCRIPTION: Construct a statespace model from a plist description
0006 %
0007 % CALL:          see ssm
0008 %
0009 % PARAMETER:
0010 %
0011 % HISTORY:       15-04-2008 Adrien
0012 %                Creation
0013 %
0014 % TODO:          implement multiple i/o when subassign function is done
0015 %
0016 % VERSION :      '$Id: ssmFromDescription.m,v 1.16 2008/08/30 19:45:08 adrien Exp $';
0017 %%%%%%%%%%%%%%%
0018 function sys = ssmFromDescription(pli)
0019     utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0020
0021 % Get default params
0022 pl = combine(pli, ssm.getDefaultPlist('From Description'));
0023 if ~isa(pl, 'plist') % checking input type
0024     error(['error because input is not a plist but a ', class(pl)]);
0025 end
0026
0027 sys(length(pl)) = ssm;
0028 for i_pl = 1:length(pl) % going through the input
0029     % filling compulsory user defined fields
0030     userfields = {'name', 'ssnames', 'outputnames', 'inputnames', 'timestep', 'amats', ...
0031                 'bmats', 'cmats', 'dmats'};
0032     % other optional fields that may be used defined
0033     otherfields = {'mmats', 'amats_handles', 'bmats_handles', 'cmats_handles', 'dmats_handles' ...
0034                  'inputvarnames', 'ssvarnames', 'outputvarnames' ...
0035                  'paramnames', 'paramvalues', 'paramsigmas'};
0036
0037     for f = userfields
0038         if isparam(pl(i_pl), f{1})
0039             sys(i_pl).(f{1}) = find(pl(i_pl), f{1});
0040         else
0041             display(['### ERROR : field in ssm named ''', f{1}, '' must be user defined ###']);
0042             display(['### list of other compulsory user defined fields ###']);
0043             display(char(userfields))
0044             display(['### list of other optional user defined fields ###']);
0045             display(char(otherfields))
0046             error(['see above message and lists ^^ ']);
0047         end
0048     end
0049
0050     for f = otherfields
0051         if isparam(pl(i_pl), f{1})
0052             sys(i_pl).(f{1}) = find(pl(i_pl), f{1});
0053         end
0054     end
0055     % fields that do not depend on input variables.
0056     sys(i_pl).addHistory(ssm.getInfo('ssm', 'From Description'), pli(i_pl), {''}, []);
0057
0058 end
0059 end

```

ssmFromFilename

PURPOSE ^

%%%

SYNOPSIS ^

```
function sys = ssmFromFilename(pli)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      ssmFromFilename
DESCRIPTION:   Construct a statespace model from a file
CALL:         see ssm
PARAMETER:
HISTORY:      15-04-2008 Hewitson
               Creation
VERSION :     '$Id: ssmFromFilename.m,v 1.11 2008/08/08 15:32:33 adrien Exp $';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      ssmFromFilename
0004 %
0005 % DESCRIPTION:   Construct a statespace model from a file
0006 %
0007 % CALL:         see ssm
0008 %
0009 % PARAMETER:
```

Description of ssmFromFilename

```
0010 %
0011 % HISTORY:      15-04-2008 Hewitson
0012 %              Creation
0013 % VERSION :     '$Id: ssmFromFilename.m,v 1.11 2008/08/08 15:32:33 adrien Exp $';
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 function sys = ssmFromFilename(pli)
0017     utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0018
0019 % Check the filename and extension
0020 filename = find(pli, 'filename');
0021 [pathstr, f_name, ext] = fileparts(filename);
0022
0023 % Add default values
0024 switch ext
0025     case '.mat'
0026         in = load(filename);
0027         sys = in.objs;
0028         sys.addHistory(ssm.getInfo('ssm', 'From MAT File'), pli, {''}, []);
0029     case '.xml'
0030         root_node = xmlread(filename);
0031         sys = ltpda_xmlread(root_node, 'ssm');
0032         sys.addHistory(ssm.getInfo('ssm', 'From XML File'), pli, {''}, []);
0033     otherwise
0034         error('### Unknown file extension.');
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ssmFromMiir

PURPOSE ^

%%%

SYNOPSIS ^

function varargout = ssmFromMiir(varargin)

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      ssmFromMiir
DESCRIPTION: Construct a statespace model from a miir
CALL:         see ssm
PARAMETER:

HISTORY:      15-04-2008 Adrien
               Creation

TODO:         check must be made there is no pole zero cancelation
VERSION :     '$Id:$';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ssm](#) SSM statespace model class constructor.

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
```

```

0003 % FUNCTION:      ssmFromMiir
0004 %
0005 % DESCRIPTION: Construct a statespace model from a miir
0006 %
0007 % CALL:           see ssm
0008 %
0009 % PARAMETER:
0010 %
0011 %
0012 % HISTORY:        15-04-2008 Adrien
0013 %                  Creation
0014 %
0015 % TODO:           check must be made there is no pole zero cancelation
0016 %
0017 % VERSION :       '$Id:$';
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020 function varargout = ssmFromMiir(varargin)
0021 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0022
0023 miirsin = varargin{1};
0024 ssmout(length(miirsin)) = ssm();
0025 for i =1:length(miirsin)
0026     num = miirsin(i).a;
0027     den = miirsin(i).b;
0028     num = num/den(1);
0029     den = den/den(1);
0030     Nss = size(den,2)-1;
0031     if length(num)<Nss+1
0032         num = [zeros(1,Nss+1-length(num)) num];
0033     end
0034     [q,r] = deconv(num,den);%polynomial division for den = conv(num,q)+r .
0035     if ~length(q)==1
0036         error('system may be non caussal');
0037     end
0038
0039     ssmout(i).dmats = {q};
0040     ssmout(i).amats = {[zeros(Nss-1,1) eye(Nss-1); fliplr(-den(2:(Nss+1)))]}];
0041     ssmout(i).bmats = {[zeros(Nss-1,1) ;1]}];
0042     ssmout(i).cmats = {r(2:(Nss+1))};
0043
0044     ssmout(i).name = miirsin(i).name;
0045     ssmout(i).timestep = 1/(miirsin(i).fs);
0046     ssmout(i).inputnames = {'U'};
0047     ssmout(i).outputnames = {'Y'};
0048     ssmout(i).addHistory(ssm.getInfo('ssm', 'From Miir'), plist(), {''}, miirsin(i).hist);
0049 end
0050
0051 varargout = {ssmout};
0052 end
0053
0054

```

ssmFromPzmodel

PURPOSE ^

%%%

SYNOPSIS ^

```
function ssmout = ssmFromPzmodel(pzmodelsin)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      ssmFromPzmodel
DESCRIPTION: Construct a statespace model from a pzmodel
CALL:         see ssm
PARAMETER:
HISTORY:      15-04-2008 Adrien
               Creation
TODO:
VERSION :     '$Id: test_ssm_constructor.m,v 1.7 2008/06/13 22:36:18 adrien Exp $';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ssm](#) SSM statespace model class constructor.

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      ssmFromPzmodel
```



```

0004 %
0005 % DESCRIPTION: Construct a statespace model from a pzmodel
0006 %
0007 % CALL:          see ssm
0008 %
0009 % PARAMETER:
0010 %
0011 % HISTORY:       15-04-2008 Adrien
0012 %                Creation
0013 %
0014 % TODO:
0015 %
0016 % VERSION :      '$Id: test_ssm_constructor.m,v 1.7 2008/06/13 22:36:18 adrien Exp $';
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 function ssmout = ssmFromPzmodel(pzmodelsin)
0020 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0021
0022 ssmout(1,length(pzmodelsin))=ssm;
0023 for i_pzms=1:length(pzmodelsin)
0024     a = 1;
0025     b = 1;
0026     for i=1:length(pzmodelsin(i_pzms).poles)
0027         if isnan(pzmodelsin(i_pzms).poles(i).q)
0028             w0 = pzmodelsin(i_pzms).poles(i).f*2*pi ;
0029             a = conv(a,[1 -w0]);
0030         else
0031             q = pzmodelsin(i_pzms).poles(i).q;
0032             w0 = 2*pi*pzmodelsin(i_pzms).poles(i).f;
0033             p = [1, -1/q*w0 w0^2];
0034             a = conv(a,p);
0035         end
0036     end
0037     for i=1:length(pzmodelsin(i_pzms).zeros)
0038         if isnan(pzmodelsin(i_pzms).zeros(i).q)
0039             w0 = pzmodelsin(i_pzms).zeros(i).f*2*pi ;
0040             b = conv(b,[1 -w0]);
0041         else
0042             q = pzmodelsin(i_pzms).zeros(i).q;
0043             w0 = 2*pi*pzmodelsin(i_pzms).zeros(i).f;
0044             p = [1, -1/q*w0 w0^2];
0045             b = conv(b,p);
0046         end
0047     end
0048     ssmout = ssm();
0049
0050     num = b*pzmodelsin(i_pzms).gain;
0051     den = a;
0052     Nss = length(den)-1;
0053     if length(num)<Nss+1
0054         num = [zeros(1,Nss+1-length(num)) num];
0055     end
0056     [q,r] = deconv(num,den);%polynomial division for den = conv(num,q)+r .
0057     if ~length(q)==1
0058         error('system may be non caussal');
0059     end
0060
0061     ssmout(i_pzms).dmats = {q};
0062     ssmout(i_pzms).amats = {[zeros(Nss-1,1) eye(Nss-1); flip1r(-den(2:(Nss+1)))]}];
0063     ssmout(i_pzms).bmats = {[zeros(Nss-1,1) ;1]}];
0064     ssmout(i_pzms).cmats = {r(2:(Nss+1))};
0065
0066     ssmout(i_pzms).name = pzmodelsin(i_pzms).name;
0067     ssmout(i_pzms).timestep = 0;
0068     ssmout(i_pzms).inputnames = {'U'};
0069     ssmout(i_pzms).outputnames = {'Y'};

```

Description of ssmFromPzmodel

```
0070    ssmout(i_pzms).addHistory(ssm.getInfo('ssm', 'From Pzmodel'), plist(), {''}, pzmodelsin(i_pzms).hist);
0071  end
0072
0073
0074  end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ssmFromRepository

PURPOSE ^

%%%

SYNOPSIS ^

```
function ss = ssmFromRepository(pli)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      ssmFromRepository
DESCRIPTION: Construct a statespace model from a repository
CALL:         see ssm
PARAMETER:
HISTORY:      15-04-2008 Hewitson
              Creation
VERSION : '$Id: ssmFromRepository.m,v 1.12 2008/08/08 15:32:33 adrien Exp $';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      ssmFromRepository
0004 %
0005 % DESCRIPTION: Construct a statespace model from a repository
0006 %
0007 % CALL:         see ssm
0008 %
```

```

0009 % PARAMETER:
0010 %
0011 % HISTORY:      15-04-2008 Hewitson
0012 %              Creation
0013 %
0014 % VERSION : '$Id: ssmFromRepository.m,v 1.12 2008/08/08 15:32:33 adrien Exp $';
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function ss = ssmFromRepository(pli)
0018 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0019
0020 % Add default values
0021 pl = combine(pli, ssm.getDefaultPlist('From Repository'));
0022
0023 % Get parameters
0024 conn = find(pl, 'conn');
0025 hostname = find(pl, 'hostname');
0026 database = find(pl, 'database');
0027 ids = find(pl, 'id');
0028
0029 % do we have a connection?
0030 closeConn = 0;
0031 if isempty(conn)
0032     closeConn = 1;
0033     % Connect to repository
0034     conn = mysql_connect(hostname, database);
0035 end
0036 if ~isa(conn, 'database')
0037     error('### connection failed. ');
0038 end
0039
0040 % Get each ID
0041 Nids = length(ids);
0042 ss = [];
0043 for kk=1:Nids
0044
0045     %---- This id
0046     id = ids(kk);
0047     disp(sprintf(' - retrieving ID %d', id));
0048
0049     %---- check ID object type
0050     tt = mysql_getObjType(conn, id);
0051     %---- If this is an AO
0052     if strcmp(tt, mfilename)
0053         %---- call database constructor
0054         s = ltpda_obj_retrieve(conn, id);
0055         %---- Add history
0056         s.addHistory(ssm.getInfo('ssm', 'From Repository'), pli, {''}, []);
0057         %---- Add to output array
0058         ss = [ss s]; %#ok<AGROW>
0059     else
0060         warning(' !skipping ID %d, type %s', id, tt);
0061     end
0062 end
0063
0064 % close connection
0065 if closeConn
0066     close(conn);
0067 end
0068
0069 end
0070

```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.2 2008/08/08 15:32:33 adrien Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.2 2008/08/08 15:32:33 adrien Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0020
0021 obj_struct = varargin{1};
0022 struct_ver = varargin{2};
0023
0024 % get the version of the current toolbox
0025 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0026
0027 % get only the version string without the MATLAB version
0028 struct_ver = strtok(struct_ver);
```

```

0029
0030 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0031
0032 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0033     strcmp(struct_ver, '1.0')
0034
0035     try
0036
0037         % Update here the structure
0038
0039     catch ME
0040         disp(varargin{1});
0041         throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (ssm) above is not from
version 1.0')));
0042     end
0043
0044     % Set the struct_ver to '1.1'
0045     struct_ver = '1.1';
0046
0047 end
0048
0049 %%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%
0050
0051 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0052     strcmp(struct_ver, '1.1')
0053
0054     % Set the struct_ver to '1.2'
0055     struct_ver = '1.1';
0056
0057 end
0058
0059 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0060
0061 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0062     strcmp(struct_ver, '1.2')
0063
0064     % Set the struct_ver to '1.3'
0065     struct_ver = '1.3';
0066
0067 end
0068
0069 varargout{1} = obj_struct;
0070

```

validate

PURPOSE ^

%%%

SYNOPSIS ^

function varargout = validate(varargin)

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      validate
DESCRIPTION:   Completes and checks content a ssm object
CALL:         ssmout = ssmFromPlist(ssmin)
               This function is private. To check and copy an object use
               obj = ssm(old) isnstead

NOTE: This is a transparent method and doesn't add history.

HISTORY:      15-04-2008 Adrien
               Creation

TODO:         possibly a mex file...?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% starting initial checks
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of ssm objects depending of the second input

This function is called by:

-  [assemble](#) assembles embedded subsystems, with exogenous inputs
-  [minreal](#) minreal gives a minimal realization of a ssm object by deleting state
-  [modifparams](#) modifparams enables to options(i_options)y and substitute parameters
-  [modify](#) modify allows to exectue a string to modify a ssm object
-  [reduce](#) reduce enables to do model simplification
-  [ssm](#) SSM statespace model class constructor.

SOURCE CODE ^

```

0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      validate
0004 %
0005 % DESCRIPTION:  Completes and checks content a ssm object
0006 %
0007 % CALL:         ssmout = ssmFromPlist(ssmin)
0008 %               This function is private. To check and copy an object use
0009 %               obj = ssm(old) instead
0010 %
0011 % NOTE: This is a transparent method and doesn't add history.
0012 %
0013 % HISTORY:      15-04-2008 Adrien
0014 %               Creation
0015 %
0016 % TODO:         possibly a mex file...?
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 function varargout = validate(varargin)
0020 %% starting initial checks
0021 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0022
0023 % Collect all SSMs
0024 sys = utils.helper.collect_objects(varargin(:), 'ssm');
0025
0026 % Decide on a deep copy or a modify, depending on the output
0027 sys = copy(sys, nargout);
0028
0029 %% begin function body
0030
0031 for i_sys = 1:length(sys) % going through the input
0032
0033     %% ===== checking no compulsory field is missing =====
0034
0035     % calculating fields length for Ninputs, Inputsizes, sssize, outputsizes,
0036     % Nparams and checking compulsory fields are well defined
0037
0038     %% ===== generating counter fields =====
0039     sys(i_sys).Ninputs = size(sys(i_sys).bmats, 2);
0040     sys(i_sys).Nss     = size(sys(i_sys).amats, 2);
0041     sys(i_sys).Noutputs = size(sys(i_sys).cmats, 1);
0042     sys(i_sys).Nparams  = length(sys(i_sys).paramnames);
0043     % generating inputsizes if it does not already exist
0044     if isempty(sys(i_sys).inputsizes)
0045         sys(i_sys).inputsizes = - ones(1, sys(i_sys).Ninputs);
0046         % has to look in every b or d matrix to find a non-empty cell
0047         for i=1:sys(i_sys).Ninputs
0048             for j=1:sys(i_sys).Nss
0049                 if ~isequal(sys(i_sys).bmats{j,i}, [])
0050                     sys(i_sys).inputsizes(i) = size(sys(i_sys).bmats{j,i},2);
0051                     break
0052                 end
0053             end
0054             for j=1:sys(i_sys).Noutputs
0055                 if ~isequal(sys(i_sys).dmats{j,i}, [])
0056                     sys(i_sys).inputsizes(i) = size(sys(i_sys).dmats{j,i},2);
0057                     break
0058                 end
0059             end
0060         end
0061     end

```



```

0062 % generating sssizes
0063 if isempty(sys(i_sys).sssizes)
0064     sys(i_sys).sssizes = - ones(1,sys(i_sys).Nss);
0065     % has to look in every b or d matrix to find a non-empty cell
0066     for i=1:sys(i_sys).Nss
0067         sys(i_sys).sssizes(i) = size(sys(i_sys).amats{i,i},2); % to modify later on
0068     end
0069 end
0070 % generating outputsizes if it does not already exist
0071 if isempty(sys(i_sys).outputsizes)
0072     sys(i_sys).outputsizes = - ones(1,sys(i_sys).Noutputs);
0073     % has to look in every b or d matrix to find a non-empty cell
0074     for i=1:sys(i_sys).Noutputs
0075         for j=1:sys(i_sys).Nss
0076             if ~isequal(sys(i_sys).cmats{i,j}, [])
0077                 sys(i_sys).outputsizes(i) = size(sys(i_sys).cmats{i,j},1);
0078                 break
0079             end
0080         end
0081         for j=1:sys(i_sys).Ninputs
0082             if ~isequal(sys(i_sys).dmats{i,j}, [])
0083                 sys(i_sys).outputsizes(i) = size(sys(i_sys).dmats{i,j},1);
0084                 break
0085             end
0086         end
0087     end
0088 end
0089
0090 %% ===== generating missing user defined fields =====
0091
0092 % data with linear system matrices
0093 % generating mmats if it does not already exist
0094 if isempty(sys(i_sys).mmats)
0095     sys(i_sys).mmats = cell(sys(i_sys).Nss,sys(i_sys).Nss);
0096     for i=1:sys(i_sys).Nss
0097         sys(i_sys).mmats{i,i} = eye(sys(i_sys).sssizes(i));
0098     end
0099 end
0100 % generating isnumerical
0101 sys(i_sys).isnumerical = true;
0102 for i=1:length(sys(i_sys).amats)
0103     if ~isreal(sys(i_sys).amats{i})
0104         sys(i_sys).isnumerical = false;
0105     end
0106 end
0107 if sys(i_sys).isnumerical
0108     for i=1:length(sys(i_sys).bmats)
0109         if ~isreal(sys(i_sys).bmats{i})
0110             sys(i_sys).isnumerical = false;
0111         end
0112     end
0113 end
0114 if sys(i_sys).isnumerical
0115     for i=1:length(sys(i_sys).cmats)
0116         if ~isreal(sys(i_sys).cmats{i})
0117             sys(i_sys).isnumerical = false;
0118         end
0119     end
0120 end
0121 if sys(i_sys).isnumerical
0122     for i=1:length(sys(i_sys).dmats)
0123         if ~isreal(sys(i_sys).dmats{i})
0124             sys(i_sys).isnumerical = false;
0125         end
0126     end
0127 end

```

```

0128     if sys(i_sys).isnumerical
0129         for i=1:length(sys(i_sys).mmats)
0130             if ~isreal(sys(i_sys).mmats{i})
0131                 sys(i_sys).isnumerical = false;
0132             end
0133         end
0134     end
0135
0136     % nonlinear handle fields
0137     %generating amats_handles if it does not already exist
0138     if isempty(sys(i_sys).amats_handles)
0139         sys(i_sys).amats_handles = cell(sys(i_sys).Nss,sys(i_sys).Nss);
0140     end
0141     %generating bmats_handles if it does not already exist
0142     if isempty(sys(i_sys).bmats_handles)
0143         sys(i_sys).bmats_handles = cell(sys(i_sys).Nss,sys(i_sys).Ninputs);
0144     end
0145     %generating cmats_handles if it does not already exist
0146     if isempty(sys(i_sys).cmats_handles)
0147         sys(i_sys).cmats_handles = cell(sys(i_sys).Noutputs,sys(i_sys).Nss);
0148     end
0149     %generating dmats_handles if it does not already exist
0150     if isempty(sys(i_sys).dmats_handles)
0151         sys(i_sys).dmats_handles = cell(sys(i_sys).Noutputs,sys(i_sys).Ninputs);
0152     end
0153
0154     % input variables array data
0155     % generating inputnames if it does not already exist
0156     if isempty(sys(i_sys).inputnames)
0157         sys(i_sys).inputnames = cell(1,sys(i_sys).Ninputs);
0158         for i=1:sys(i_sys).Ninputs
0159             sys(i_sys).inputnames{i} = ['input ',utils.helper.num2str(i)];
0160         end
0161     end
0162     % generating inputvarnames if it does not already exist
0163     if isempty(sys(i_sys).inputvarnames)
0164         sys(i_sys).inputvarnames = cell(1,sys(i_sys).Ninputs);
0165         for i=1:sys(i_sys).Ninputs
0166             sys(i_sys).inputvarnames{i} = cell(1,sys(i_sys).inputsizes(i));
0167             for j=1:sys(i_sys).inputsizes(i)
0168                 sys(i_sys).inputvarnames{i}{j} = [sys(i_sys).inputnames{i}, '>' utils.helper.num2str(j)];
0169             end
0170         end
0171     end
0172
0173     % state space variables array data
0174     % generating ssnames if it does not already exist
0175     if isempty(sys(i_sys).ssnames)
0176         sys(i_sys).ssnames = cell(1,sys(i_sys).Nss);
0177         for i=1:sys(i_sys).Nss
0178             sys(i_sys).ssnames{i} = ['state ',utils.helper.num2str(i)];
0179         end
0180     end
0181     % generating ssvarnames if it does not already exist
0182     if isempty(sys(i_sys).ssvarnames)
0183         sys(i_sys).ssvarnames = cell(1,sys(i_sys).Nss);
0184         for i=1:sys(i_sys).Nss
0185             sys(i_sys).ssvarnames{i} = cell(1,sys(i_sys).sssizes(i));
0186             for j=1:sys(i_sys).sssizes(i)
0187                 sys(i_sys).ssvarnames{i}{j} = [sys(i_sys).ssnames{i}, '>' utils.helper.num2str(j)];
0188             end
0189         end
0190     end
0191
0192     %for outputs
0193     % generating outputnames if it does not already exist

```

```

0194 if isempty(sys(i_sys).outputnames)
0195     sys(i_sys).outputnames = cell(1,sys(i_sys).Noutputs);
0196     for i=1:sys(i_sys).Noutputs
0197         sys(i_sys).outputnames{i} = ['output ',utils.helper.num2str(i)];
0198     end
0199 end
0200 % generating outputvarnames if it does not already exist
0201 if isempty(sys(i_sys).outputvarnames)
0202     sys(i_sys).outputvarnames = cell(1,sys(i_sys).Noutputs);
0203     for i=1:sys(i_sys).Noutputs
0204         sys(i_sys).outputvarnames{i} = cell(1,sys(i_sys).outputsizes(i));
0205         for j=1:sys(i_sys).outputsizes(i)
0206             sys(i_sys).outputvarnames{i}{j} = [sys(i_sys).outputnames{i}, '>' utils.helper.num2str(j)];
0207         end
0208     end
0209 end
0210
0211 % parameter fields
0212 % generating paramsigmas if it does not already exist
0213 if isempty(sys(i_sys).paramsigmas)
0214     sys(i_sys).paramsigmas = zeros(1,sys(i_sys).Nparams);
0215 end
0216
0217 % generating paramvalues if it does not already exist
0218 if isempty(sys(i_sys).paramvalues)
0219     sys(i_sys).paramvalues = zeros(1,sys(i_sys).Nparams);
0220 end
0221
0222 %% ===== checking field sizes =====
0223 % checking compatibility with Ninputs
0224 if ~(
0225     sys(i_sys).Ninputs==size(sys(i_sys).bmats,2) ...
0226     && sys(i_sys).Ninputs==size(sys(i_sys).dmats,2) ...
0227     && sys(i_sys).Ninputs==size(sys(i_sys).bmats_handles,2) ...
0228     && sys(i_sys).Ninputs==size(sys(i_sys).dmats_handles,2) ...
0229     && sys(i_sys).Ninputs==length(sys(i_sys).inputnames) ...
0230     && sys(i_sys).Ninputs==length(sys(i_sys).inputvarnames) ...
0231     && sys(i_sys).Ninputs==length(sys(i_sys).inputsizes) )
0232     error(['error in ssm ',sys(i_sys).name,' because there are ',...
0233         num2str(size(sys(i_sys).bmats,2)), ' columns in B , ',...
0234         num2str(size(sys(i_sys).dmats,2)), ' columns in D , ',...
0235         num2str(size(sys(i_sys).bmats_handles,2)), ' columns in bmats_handles , ',...
0236         num2str(size(sys(i_sys).dmats_handles,2)), ' columns in dmats_handles , ',...
0237         num2str(length(sys(i_sys).inputnames)), ' inputnames, ',...
0238         num2str(length(sys(i_sys).inputvarnames)), ' blocks in inputvarnames and ',...
0239         num2str(length(sys(i_sys).inputsizes)), ' inputsizes' ] );
0240 end
0241
0242 % checking compatibility with Nss
0243 if ~(
0244     sys(i_sys).Nss==size(sys(i_sys).amats,2) ...
0245     && sys(i_sys).Nss==size(sys(i_sys).amats,1) ...
0246     && sys(i_sys).Nss==size(sys(i_sys).bmats,1) ...
0247     && sys(i_sys).Nss==size(sys(i_sys).cmats,2) ...
0248     && sys(i_sys).Nss==size(sys(i_sys).amats_handles,2) ...
0249     && sys(i_sys).Nss==size(sys(i_sys).amats_handles,1) ...
0250     && sys(i_sys).Nss==size(sys(i_sys).cmats_handles,2) ...
0251     && sys(i_sys).Nss==length(sys(i_sys).ssnames) ...
0252     && sys(i_sys).Nss==length(sys(i_sys).ssvarnames) ...
0253     && sys(i_sys).Nss==length(sys(i_sys).sssizes) )
0254     error(['error in ssm ',sys(i_sys).name,' because there are ',...
0255         num2str(size(sys(i_sys).amats,2)), ' columns in A , ',...
0256         num2str(size(sys(i_sys).amats,1)), ' lines in A , ',...
0257         num2str(size(sys(i_sys).bmats,1)), ' lines in B , ',...
0258         num2str(size(sys(i_sys).cmats,2)), ' columns in C , ',...
0259         num2str(size(sys(i_sys).amats_handles,2)), ' columns in A_handles , ',...

```

```

0260     num2str(size(sys(i_sys).amats_handles,1)), ' lines in A_handles, ', ...
0261     num2str(size(sys(i_sys).bmats_handles,1)), ' lines in B_handles, ', ...
0262     num2str(size(sys(i_sys).cmats_handles,2)), ' columns in C_handles, ', ...
0263     num2str(length(sys(i_sys).ssnames)), ' ssnames, ', ...
0264     num2str(length(sys(i_sys).ssvarnames)), ' blocks in ssvarnames and ', ...
0265     num2str(length(sys(i_sys).sssizes)), ' sssizes' ] );
0266 end
0267
0268 % checking compatibility with Noutputs
0269 if ~( sys(i_sys).Noutputs==size(sys(i_sys).cmats,1) ...
0270     && sys(i_sys).Noutputs==size(sys(i_sys).dmats,1) ...
0271     && sys(i_sys).Noutputs==size(sys(i_sys).cmats_handles,1) ...
0272     && sys(i_sys).Noutputs==size(sys(i_sys).dmats_handles,1) ...
0273     && sys(i_sys).Noutputs==length(sys(i_sys).outputnames) ...
0274     && sys(i_sys).Noutputs==length(sys(i_sys).outputvarnames) ...
0275     && sys(i_sys).Noutputs==length(sys(i_sys).outputsizes) )
0276     error(['error in ssm ', sys(i_sys).name, ' because there are ', ...
0277         num2str(size(sys(i_sys).cmats,1)), ' lines in C, ', ...
0278         num2str(size(sys(i_sys).dmats,1)), ' lines in D, ', ...
0279         num2str(size(sys(i_sys).cmats_handles,1)), ' lines in C_handles, ', ...
0280         num2str(size(sys(i_sys).dmats_handles,1)), ' lines in D_handles, ', ...
0281         num2str(length(sys(i_sys).outputnames)), ' outputnames, ', ...
0282         num2str(length(sys(i_sys).outputvarnames)), ' blocks in outputvarnames and ', ...
0283         num2str(length(sys(i_sys).outputsizes)), ' outputsizes' ] );
0284 end
0285
0286 % Checking compatibility with Inputsizes
0287 for i=1:sys(i_sys).Ninputs
0288     if ~isequal(sys(i_sys).inputsizes(i), -1)
0289         if ~( sys(i_sys).inputsizes(i) == length(sys(i_sys).inputvarnames{i}) )
0290             error(['error in ssm ', sys(i_sys).name, ...
0291                 ' because the input number ', num2str(i), ...
0292                 ' named ', sys(i_sys).inputnames{i}, ...
0293                 ' and of size ', num2str(sys(i_sys).inputsizes(i)), ...
0294                 ' has an inputvarnames vector of size ', num2str(length(sys(i_sys).inputvarnames{i})) ] );
0295         end
0296     else
0297         str=['warning, input named ', sys(i_sys).inputnames{i}, ...
0298             ' has all matrices empty, should be deleted' ] ;
0299         utils.helper.msg(utils.const.msg.MNAME, str);
0300     end
0301     for j=1:sys(i_sys).Noutputs
0302         if ~isequal(sys(i_sys).dmats{j,i}, [])
0303             if ~( sys(i_sys).inputsizes(i) == size(sys(i_sys).dmats{j,i},2) )
0304                 error(['error in ssm ', sys(i_sys).name, ...
0305                     ' because the input number ', num2str(i), ...
0306                     ' named ', sys(i_sys).inputnames{i}, ...
0307                     ' and of size ', num2str(sys(i_sys).inputsizes(i)), ...
0308                     ' has a D matrix of width ', num2str(size(sys(i_sys).dmats{j,i},2)) ] );
0309             end
0310         end
0311     end
0312     for j=1:sys(i_sys).Nss
0313         if ~isequal(sys(i_sys).bmats{j,i}, [])
0314             if ~( sys(i_sys).inputsizes(i) == size(sys(i_sys).bmats{j,i},2) )
0315                 error(['error in ssm ', sys(i_sys).name, ...
0316                     ' because the input number ', num2str(i), ...
0317                     ' named ', sys(i_sys).inputnames{i}, ...
0318                     ' and of size ', num2str(sys(i_sys).inputsizes(i)), ...
0319                     ' has a B matrix of width ', num2str(size(sys(i_sys).bmats{j,i},2)) ] );
0320             end
0321         end
0322     end
0323 end
0324
0325 % Checking compatibility with Outputsizes

```

```

0326 for i=1:sys(i_sys).Noutputs
0327     if ~isequal(sys(i_sys).outputsizes(i), -1)
0328         if ~( sys(i_sys).outputsizes(i) == length(sys(i_sys).outputvarnames{i}) )
0329             error(['error in ssm ', sys(i_sys).name, ...
0330                 ' because the output number ', num2str(i), ...
0331                 ' named ', sys(i_sys).outputnames{i}, ...
0332                 ' and of size ', num2str(sys(i_sys).outputsizes(i)), ...
0333                 ' has an outputvarnames vector of size ', num2str(length(sys(i_sys).outputvarnames{i})) ]);
0334         end
0335     else
0336         str = ['warning, output named ', sys(i_sys).outputnames{i}, ...
0337             ' has all matrices empty, should be deleted'];
0338         utils.helper.msg(utils.const.msg.MNAME, str);
0339     end
0340     for j=1:sys(i_sys).Ninputs
0341         if ~isequal(sys(i_sys).dmats{i,j}, [])
0342             if ~( sys(i_sys).outputsizes(i) == size(sys(i_sys).dmats{i,j},1) )
0343                 error(['error in ssm ', sys(i_sys).name, ...
0344                     ' because the output number ', num2str(i), ...
0345                     ' named ', sys(i_sys).outputnames{i}, ...
0346                     ' and of size ', num2str(sys(i_sys).outputsizes(i)), ...
0347                     ' has a D matrix of height ', num2str(size(sys(i_sys).dmats{i,j},1)) ]);
0348             end
0349         end
0350     end
0351     for j=1:sys(i_sys).Nss
0352         if ~isequal(sys(i_sys).cmats{i,j}, [])
0353             if ~( sys(i_sys).outputsizes(i) == size(sys(i_sys).cmats{i,j},1) )
0354                 error(['error in ssm ', sys(i_sys).name, ...
0355                     ' because the output number ', num2str(i), ...
0356                     ' named ', sys(i_sys).outputnames{i}, ...
0357                     ' and of size ', num2str(sys(i_sys).outputsizes(i)), ...
0358                     ' has a C matrix of height ', num2str(size(sys(i_sys).cmats{i,j},1)) ]);
0359             end
0360         end
0361     end
0362 end
0363
0364 % Checking compatibility with sssizes
0365 for i=1:sys(i_sys).Nss
0366     if ~isequal(sys(i_sys).sssizes(i), -1)
0367         if ~( sys(i_sys).sssizes(i) == length(sys(i_sys).ssvarnames{i}) )
0368             error(['error in ssm ', sys(i_sys).name, ...
0369                 ' because the state space number ', num2str(i), ...
0370                 ' named ', sys(i_sys).ssnames{i}, ...
0371                 ' and of size ', num2str(sys(i_sys).sssizes(i)), ...
0372                 ' has a ssvarnames vector of size ', num2str(length(sys(i_sys).ssvarnames{i})) ]);
0373         end
0374     else
0375         str = ['warning, output named ', sys(i_sys).outputnames{i}, ...
0376             ' has all matrices empty, should be deleted'];
0377         utils.helper.msg(utils.const.msg.MNAME, str);
0378     end
0379     for j=1:sys(i_sys).Ninputs
0380         if ~isequal(sys(i_sys).bmats{i,j}, [])
0381             if ~( sys(i_sys).sssizes(i) == size(sys(i_sys).bmats{i,j},1) )
0382                 error(['error in ssm ', sys(i_sys).name, ...
0383                     ' because the state space number ', num2str(i), ...
0384                     ' named ', sys(i_sys).ssnames{i}, ...
0385                     ' and of size ', num2str(sys(i_sys).sssizes(i)), ...
0386                     ' has a B matrix of height ', num2str(size(sys(i_sys).bmats{i,j},1)) ]);
0387             end
0388         end
0389     end
0390     for j=1:sys(i_sys).Nss
0391         if ~isequal(sys(i_sys).mmats{i,j}, [])

```

```

0392     if ~( sys(i_sys).sssizes(i) == size(sys(i_sys).mmats{i,j},1) ...
0393         && sys(i_sys).sssizes(j) == size(sys(i_sys).mmats{i,j},2) )
0394         error(['error in ssm ', sys(i_sys).name, ...
0395             ' because the state space position ', num2str(i), ' ', num2str(j), ...
0396             ' named ', sys(i_sys).ssnames{i}, ' and ', sys(i_sys).ssnames{j}, ...
0397             ' and of size ', num2str([sys(i_sys).sssizes(i),sys(i_sys).sssizes(j)] ), ...
0398             ' has a A matrix of size ', num2str(size(sys(i_sys).amats{i,j})) ]]);
0399     end
0400 end
0401 end
0402 for j=1:sys(i_sys).Noutputs
0403     if ~isequal(sys(i_sys).cmats{j,i}, [])
0404         if ~( sys(i_sys).sssizes(i) == size(sys(i_sys).cmats{j,i},2) )
0405             error(['error in ssm ', sys(i_sys).name, ...
0406                 ' because the state space number ', num2str(i), ...
0407                 ' named ', sys(i_sys).ssnames{i}, ...
0408                 ' and of size ', num2str(sys(i_sys).sssizes(i)), ...
0409                 ' has a C matrix of width ', num2str(size(sys(i_sys).cmats{j,i},2)) ]]);
0410         end
0411     end
0412 end
0413 end
0414
0415 % checking compatibility with Nparams
0416 if ~( sys(i_sys).Nparams == length(sys(i_sys).paramvalues) ...
0417     && sys(i_sys).Nparams == length(sys(i_sys).paramsigmas) )
0418     error(['error in ssm ', sys(i_sys).name, ' because there are ', ...
0419         num2str(sys(i_sys).Nparams), ' parameter names but ', ...
0420         num2str(length(sys(i_sys).paramvalues)), ' parameter values, and ', ...
0421         num2str(length(sys(i_sys).paramsigmas)), ' parametersigmas.']);
0422 end
0423
0424 %% ===== checking redundancies =====
0425 for i_input=1:sys(i_sys).Ninputs
0426     [pos, sum] = ssm.cellstrfind(sys(i_sys).inputnames, sys(i_sys).inputnames{i_input}, 'all');
0427     if sum>1
0428         error(['error because system ', sys(i_sys).name, ' has ', num2str(sum), ' inputs named ', sys(i_sys).inputnames{i_input}]);
0429     end
0430 end
0431 for i_output=1:sys(i_sys).Noutputs
0432     [pos, sum] = ssm.cellstrfind(sys(i_sys).outputnames, sys(i_sys).outputnames{i_output}, 'all');
0433     if sum>1
0434         error(['error because system ', sys(i_sys).name, ' has ', num2str(sum), ' outputs named ', sys(i_sys).outputnames{i_output}]);
0435     end
0436 end
0437 for i_ss=1:sys(i_sys).Nss
0438     [pos, sum] = ssm.cellstrfind(sys(i_sys).ssnames, sys(i_sys).ssnames{i_ss}, 'all');
0439     if sum>1
0440         error(['error because system ', sys(i_sys).name, ' has ', num2str(sum), ' state space named ', sys(i_sys).ssnames{i_ss}]);
0441     end
0442 end
0443 for i_input=1:sys(i_sys).Nparams
0444     if i_input <= sys(i_sys).Nparams
0445         [pos, sum] = ssm.cellstrfind(sys(i_sys).paramnames, sys(i_sys).paramnames{i_input}, 'all');
0446         if sum>1
0447             if sys(i_sys).paramvalues(pos(1))==sys(i_sys).paramvalues(pos(1))
0448                 if sys(i_sys).paramsigmas(pos(1))==sys(i_sys).paramsigmas(pos(1))
0449                     sys(i_sys).paramsigmas = [sys(i_sys).paramsigmas(1:pos(1)-1) sys(i_sys).paramsigmas(pos(1)+1 : sys(i_sys).Nparams)];
0450                     sys(i_sys).paramvalues = [sys(i_sys).paramvalues(1:pos(1)-1) sys(i_sys).paramvalues(pos(1)+1 : sys(i_sys).Nparams)];
0451                     sys(i_sys).paramnames = [sys(i_sys).paramnames(1:pos(1)-1) sys(i_sys).paramnames(pos(1)+1 : sys(i_sys).Nparams)];
0452                     sys(i_sys).Nparams = sys(i_sys).Nparams-1;
0453                     str = ['warning because parameter ', sys(i_sys).paramnames{pos(2)-1}, ' was found ', num2str(sum), ' times in ',
0454                         sys(i_sys).name];
0455                     utils.helper.msg(utils.const.msg.MNAME, str);
0456                 else
0457                     str = ['error because system ', sys(i_sys).name, ' has ', num2str(sum), ' parameters named

```



```

0457 sys(i_sys).paramnames{i_input}, 'with different sigmas'];
0458     utils.helper.msg(utils.const.msg.MNAME, str);
0459     end
0460     else
0461         str = ['error because system ',sys(i_sys).name, ' has ',num2str(sum),' parameters named ',sys(i_sys).paramnames{i_input},
0462             'with different values'];
0463         utils.helper.msg(utils.const.msg.MNAME, str);
0464     end
0465 end
0466
0467 %% ===== checking field types =====
0468
0469 Fields = {...
0470     'amats','bmats' 'cmats' 'dmats' 'mmats' 'isnumerical' 'timestep' ...
0471     'amats_handles' 'bmats_handles' 'cmats_handles' 'dmats_handles' ...
0472     'inputnames' 'inputvarnames' 'inputsizes' 'Ninputs' ...
0473     'ssnames' 'ssvarnames' 'sssizes' 'Nss' ...
0474     'outputnames' 'outputvarnames' 'outputsizes' 'Noutputs' ...
0475     'paramnames' , 'paramvalues' 'paramsigmas' 'Nparams' };
0476
0477 Classes = {...
0478     'doublesym' 'doublesym' 'doublesym' 'doublesym' 'doublesym' 'logical' 'double' ...
0479     'cell' 'cell' 'cell' 'cell' ...
0480     'cellstr' 'cellcellstr' 'double' 'double' ...
0481     'cellstr' 'cellcellstr' 'double' 'double' ...
0482     'cellstr' 'double' 'double' 'double' ...
0483 };
0484
0485 for f = [Fields ; Classes]
0486     if isequal(f{2},'doublesym') % case where both double and symbolic classes are allowed
0487         for i_input = 1:length(sys(i_sys).(f{1}))
0488             if ~( isa(sys(i_sys).(f{1})){i_input},'double') || isa(sys(i_sys).(f{1})){i_input},'sym' )
0489                 error(['error because in ssm ',sys(i_sys).name, ' because element ',num2str(i_input), ...
0490                     ' of field ', f{1}, ' is of type ',class(sys(i_sys).(f{1})), ...
0491                     ' instead of '' double'' or ''sym'' ']);
0492             end
0493         end
0494     elseif isequal(f{2},'cellstr')
0495         if ~( iscellstr(sys(i_sys).(f{1})))
0496             error(['error because in ssm ',sys(i_sys).name, ...
0497                 ' because field ', f{1}, ' is of type ',class(sys(i_sys).(f{1})), ...
0498                 ' instead of ''cellstr'' ']);
0499         end
0500     elseif isequal(f{2},'cellcellstr')
0501         if ~( iscell(sys(i_sys).(f{1})))
0502             error(['error because in ssm ',sys(i_sys).name, ...
0503                 ' because field ', f{1}, ' is of type ',class(sys(i_sys).(f{1})), ...
0504                 ' instead of ''cellcellstr'' ']);
0505         end
0506         for i=1:length(sys(i_sys).(f{1}))
0507             if ~( iscellstr(sys(i_sys).(f{1})){i})
0508                 error(['error because in ssm ',sys(i_sys).name, ...
0509                     ' because field ', f{1}, ' is of type ',class(sys(i_sys).(f{1})), ...
0510                     ' instead of ''cellcellstr'' ']);
0511             end
0512         end
0513     else
0514         if ~isa(sys(i_sys).(f{1}),f{2})
0515             error(['error because in ssm ',sys(i_sys).name, ' because field ', ...
0516                 f{1}, ' is of type ',class(sys(i_sys).(f{1})), ' instead of ', f{2}]);
0517         end
0518     end
0519 end
0520 end

```

Description of validate

```
0521     if nargout > 0
0522         varargout = {sys};
0523     end
0524 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ssmFromss

PURPOSE ^

%%%

SYNOPSIS ^

```
function sys = ssmFromss(ss)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      ssmFromss
DESCRIPTION: Construct a ltpda statespace object from a matlab statespace
              object
CALL:          see ssm
PARAMETER:
HISTORY:      15-04-2008 Adrien
              Creation
TODO:         inplemnet multiple i/o when subassign function is done
VERSION :     '$Id: test_ssm_constructor.m,v 1.7 2008/06/13 22:36:18 adrien Exp $';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ssm](#) SSM statespace model class constructor.

This function is called by:

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      ssmFromss
0004 %
0005 % DESCRIPTION: Construct a ltpda statespace object from a matlab statespace
0006 %              object
```

```

0007 %
0008 % CALL:      see ssm
0009 %
0010 % PARAMETER:
0011 %
0012 % HISTORY:    15-04-2008 Adrien
0013 %              Creation
0014 %
0015 % TODO:        implement multiple i/o when subassign function is done
0016 %
0017 % VERSION :    '$Id: test_ssm_constructor.m,v 1.7 2008/06/13 22:36:18 adrien Exp $';
0018 %%%%%%%%%%%%%%
0019 function sys = ssmFromss(ss)
0020 utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
0021
0022 sys = ssm;
0023 sys.name = ss.Name;
0024 sys.description = ss.Notes;
0025
0026 [a,b,c,d,Ts] = ssdata(ss) ;
0027 sys.amats = {a};
0028 sys.bmats = {b};
0029 sys.cmats = {c};
0030 sys.dmats = {d};
0031 sys.timestep = Ts;
0032
0033 sys.ssname = {'ssname'};
0034 sys.inputnames = {'inputname'};
0035 sys.outputnames = {'outputname'};
0036 sys.ssvarnames = ss.StateName;
0037 sys.inputvarnames = ss.InputName;
0038 sys.outputvarnames = ss.OutputName;
0039
0040 sys.addHistory(ssm.getInfo('ssm', 'From ss'), plist, {''}, []);
0041
0042 end

```

fromA

PURPOSE ^

Construct an mfir from coefficients

SYNOPSIS ^

function f = fromA(f, pli)

DESCRIPTION ^

Construct an mfir from coefficients
%%%

FUNCTION: fromA

DESCRIPTION: Construct an mfir from coefficients

CALL: f = fromA(f, pli)

PARAMETER:

 pli: Parameter list object

HISTORY: 09-06-2008 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % Construct an mfir from coefficients
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:        fromA
0005 %
0006 % DESCRIPTION: Construct an mfir from coefficients
0007 %
0008 % CALL:            f = fromA(f, pli)
0009 %
0010 % PARAMETER:
0011 %                pli:        Parameter list object
0012 %
0013 % HISTORY:        09-06-2008 M Hewitson
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function f = fromA(f, pli)
0019
0020     VERSION = '$Id: fromA.m,v 1.7 2008/09/04 13:37:14 ingo Exp $';
0021
0022     ii = mfir.getInfo('mfir', 'From A');
0023     % Set the method version string in the minfo object
0024     ii.setMversion([VERSION '-->' ii.mversion]);
0025
0026     % Add default values
0027     pl = combine(pli, ii.plists);
0028
```

```
0029 a = find(pl, 'a');
0030 fs = find(pl, 'fs');
0031
0032 % Checking the coefficients are listed in rows
0033 if size(a,1)~=1
0034     a = a ;
0035 end
0036
0037 % Zero pad coefficients to avoid zero length history
0038 if numel(a) <= 1
0039     a = [a 0];
0040 end
0041
0042 f.setName('A', 'internal');
0043 f.setFs(fs);
0044 f.setA(a);
0045 f.setGd((f.ntaps+1)/2);
0046 f.setHistout(zeros(1,f.ntaps-1)); % initialise output history
0047
0048 % Add history
0049 f.addHistory(ii, pl, [], []);
0050
0051 end % End fromA
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromAO

PURPOSE ^

create FIR filter from magnitude of input AO/fsdata

SYNOPSIS ^

function filt = fromAO(filt, pli)

DESCRIPTION ^

```
create FIR filter from magnitude of input AO/fsdata
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      mfirFromAO

DESCRIPTION:   create FIR filter from magnitude of input AO/fsdata

CALL:         f = mfirFromAO(a, pli, version, algoname)

PARAMETER:    a:      Analysis object
               pli:    Parameter list object
               version: cvs version string
               algoname: The m-file name (use the mfilename command)

HISTORY:      22-03-2008 M Hueller
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % create FIR filter from magnitude of input AO/fsdata
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      mfirFromAO
0005 %
0006 % DESCRIPTION:   create FIR filter from magnitude of input AO/fsdata
0007 %
0008 % CALL:         f = mfirFromAO(a, pli, version, algoname)
0009 %
0010 % PARAMETER:    a:      Analysis object
0011 %               pli:    Parameter list object
0012 %               version: cvs version string
0013 %               algoname: The m-file name (use the mfilename command)
0014 %
0015 % HISTORY:      22-03-2008 M Hueller
0016 %               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 function filt = fromAO(filt, pli)
0021
0022     import utils.const.*
0023
0024     VERSION = '$Id: fromAO.m,v 1.10 2008/09/04 13:37:14 ingo Exp $';
0025     ii = mfir.getInfo('mfir', 'From AO');
0026     % Set the method version string in the minfo object
```

```

0027 ii.setMversion([VERSION '-->' ii.mversion]);
0028
0029 % Add default values
0030 pl = combine(pli, ii.plist);
0031
0032 % Get parameters
0033 a = find(pl, 'AO');
0034 N = find(pl, 'N');
0035 win = find(pl, 'win');
0036 method = find(pl, 'method');
0037
0038 % Check that a.data is a fsdata object
0039 if ~isa(a.data, 'fsdata')
0040     error('### Please use an analysis object with a fsdata data object to create a mfir
object. ');
0041 end
0042
0043 fs = a.data.fs;
0044 f = a.data.getX;
0045 xx = abs(a.data.getY);
0046
0047 ffm = f/(fs/2);
0048 switch method
0049     case 'frequency-sampling'
0050         % check window
0051         if length(win.win) ~= N+1
0052             warning('!!! resizing window function to match desired filter order. ');
0053             if strcmp(win.type, 'Kaiser') || strcmp(win.type, 'Flattop')
0054                 win = specwin(win.type, N+1, win.psl);
0055             else
0056                 win = specwin(win.type, N+1);
0057             end
0058         end
0059         utils.helper.msg(msg.OPROC2, 'designing filter using frequency-sampling method
[help fir2]');
0060         mtaps = fir2(N, ffm, xx, win.win);
0061     case 'least-squares'
0062         error('### this design method is not working properly yet. ');
0063         if mod(length(ffm), 2)
0064             ffm = ffm(1:end-1);
0065             xx = xx(1:end-1);
0066         end
0067         utils.helper.msg(msg.OPROC2, 'designing filter using least-squares method [help
firls]');
0068         mtaps = firls(N, ffm, xx);
0069     case 'Parks-McClellan'
0070         error('### this design method is not working properly yet. ');
0071         utils.helper.msg(msg.OPROC2, 'designing filter using Parks-McClellan method [help
firpm]');
0072         mtaps = firpm(N, ffm, xx);
0073     otherwise
0074         error('### unknown filter design method. ');
0075 end
0076
0077 % Make mfir object
0078 filt.setName(sprintf('fir(%s)', a.name), 'internal');
0079 filt.setFs(fs);
0080 filt.setA(mtaps);
0081 filt.setGd((filt.ntaps+1)/2);
0082 filt.setHistout(zeros(1, filt.ntaps-1));
0083
0084 % Add history
0085 filt.addHistory(ii, pl, [], []);
0086
0087 end % function f = mfirFromAO(a, pli, version, algoname)

```

fromPzmodel

PURPOSE ^

Construct an mfir from a pzmodel

SYNOPSIS ^

function f = fromPzmodel(f, pli)

DESCRIPTION ^

```
Construct an mfir from a pzmodel
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromPzmodel

DESCRIPTION: Construct an mfir from a pzmodel

CALL:         f = fromPzmodel(f, pli)

PARAMETER:    type:      String with filter type description
               pli:      Parameter list object

HISTORY:      09-06-2008 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % Construct an mfir from a pzmodel
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromPzmodel
0005 %
0006 % DESCRIPTION: Construct an mfir from a pzmodel
0007 %
0008 % CALL:         f = fromPzmodel(f, pli)
0009 %
0010 % PARAMETER:    type:      String with filter type description
0011 %               pli:      Parameter list object
0012 %
0013 % HISTORY:      09-06-2008 M Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function f = fromPzmodel(f, pli)
0019
0020     VERSION = '$Id: fromPzmodel.m,v 1.6 2008/09/04 13:37:14 ingo Exp $';
0021     ii = mfir.getInfo('mfir', 'From Pzmodel');
0022     % Set the method version string in the minfo object
0023     ii.setMversion([VERSION '-->' ii.mversion]);
0024
0025     % Add default values
0026     pl = combine(pli, ii.plist);
0027
0028     % Get parameters
```

```
0029 pzm = find(pl, 'pzmodel');
0030 fs = find(pl, 'fs');
0031
0032 if isempty(fs)
0033     % get max freq in pzmodel
0034     fs = 8*getupperFreq(pzm);
0035     warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2f Hz.', fs)...
0036             sprintf('\nThe filter will be redesigned later when used.')] );
0037 end
0038 % make MIIR filter
0039 f = tomfir(pzm, plist('fs', fs));
0040 f.setName(pzm.name, 'internal');
0041
0042 % Add history
0043 f.addHistory(ii, pl, [], []);
0044
0045 end % End fromPzmodel
```


fromStandard

PURPOSE ^

Construct an mfir from a standard types

SYNOPSIS ^

function f = fromStandard(f, pli)

DESCRIPTION ^

```
Construct an mfir from a standard types
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromStandard

DESCRIPTION: Construct an mfir from a standard types

CALL:         f = fromStandard(f, pli)

PARAMETER:    type:      String with filter type description
               pli:      Parameter list object

HISTORY:      22-03-2008 M Hueller
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [mkbandpass](#) MKBANDPASS return a bandpass filter mfir(). A Cheby filter is used.
-  [mkbandreject](#) MKBANDREJECT return a low pass filter mfir(). A Butterworth filter is used.
-  [mkhighpass](#) MKHIGHPASS return a high pass filter mfir(). A Butterworth filter is used.
-  [mklowpass](#) MKLOWPASS return a low pass filter mfir().

This function is called by:

-  [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % Construct an mfir from a standard types
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromStandard
0005 %
0006 % DESCRIPTION: Construct an mfir from a standard types
0007 %
0008 % CALL:         f = fromStandard(f, pli)
0009 %
0010 % PARAMETER:    type:      String with filter type description
0011 %               pli:      Parameter list object
0012 %
0013 % HISTORY:      22-03-2008 M Hueller
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function f = fromStandard(f, pli)
0019
0020     VERSION = '$Id: fromStandard.m,v 1.5 2008/09/04 13:37:14 ingo Exp $';
```

```
0021
0022     ii = mfir.getInfo('mfir', 'From Standard Type');
0023     % Set the method version string in the minfo object
0024     ii.setMversion([VERSION '-->' ii.mversion]);
0025
0026     % Add default values
0027     pl = combine(pli, ii.plists);
0028
0029     % Get parameters
0030     type = find(pl, 'type');
0031
0032     % check and fill parameter list
0033     plo = mfir.parseFilterParams(pl);
0034     switch type
0035     case 'lowpass'
0036         f = mklowpass(f, plo);
0037     case 'highpass'
0038         f = mkhighpass(f, plo);
0039     case 'bandpass'
0040         f = mkbandpass(f, plo);
0041     case 'bandreject'
0042         f = mkbandreject(f, plo);
0043     otherwise
0044         error('### unknown standard filter type in mfir constructor.');
```

```
0045     end
0046
0047     % Add history
0048     f.addHistory(ii, pl, [], []);
0049
0050 end % function f = mfirFromStandardType(type, pli, version, algoname)
```

mkbandpass

PURPOSE ^

MKBANDPASS return a bandpass filter mfir(). A Cheby filter is used.

SYNOPSIS ^

function f = mkbandpass(f, pl)

DESCRIPTION ^

```
MKBANDPASS return a bandpass filter mfir(). A Cheby filter is used.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MKBANDPASS return a bandpass filter mfir().
             A Cheby filter is used.

CALL:       f = mkbandpass(f, pl)

VERSION: $Id: mkbandpass.m,v 1.3 2008/08/06 15:03:49 ingo Exp $

HISTORY: 27-08-2002 M Hewitson
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [fromStandard](#) Construct an mfir from a standard types
-  [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKBANDPASS return a bandpass filter mfir(). A Cheby filter is used.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MKBANDPASS return a bandpass filter mfir().
0005 %             A Cheby filter is used.
0006 %
0007 % CALL:       f = mkbandpass(f, pl)
0008 %
0009 % VERSION: $Id: mkbandpass.m,v 1.3 2008/08/06 15:03:49 ingo Exp $
0010 %
0011 % HISTORY: 27-08-2002 M Hewitson
0012 %             Creation
0013 %
0014 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0015 %
0016 function f = mkbandpass(f, pl)
0017
0018     g      = find(pl, 'gain');
0019     fc     = find(pl, 'fc');
0020     fs     = find(pl, 'fs');
0021     order  = find(pl, 'order');
0022     win    = find(pl, 'win');
0023
0024     if(fc > fs/2)
0025         error('fc must be < fs/2');
0026     end
0027     if(fc(1) > fc(2))
0028         error('fc(1) must be < fc(2)');
```

```
0029 end
0030
0031 f.name      = 'std bandpass';
0032 f.fs        = fs;
0033 f.a         = g.*fir1(order, 2.*fc/fs, 'bandpass', win.win);
0034 f.gd        = (f.ntaps)/2;
0035 f.histout   = zeros(1,f.ntaps-1); % initialise output history
0036
0037
```

mkbandreject

PURPOSE ^

MKBANDREJECT return a low pass filter mfir(). A Butterworth filter is used.

SYNOPSIS ^

function f = mkbandreject(f, pl)

DESCRIPTION ^

MKBANDREJECT return a low pass filter mfir(). A Butterworth filter is used.
DESCRIPTION: MKBANDREJECT return a low pass filter mfir().
A Butterworth filter is used.
CALL: f = mkbandreject(f, pl)
VERSION: \$Id: mkbandreject.m,v 1.3 2008/08/06 15:03:49 ingo Exp \$
HISTORY: 27-08-2002 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  fromStandard Construct an mfir from a standard types
-  mfir MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKBANDREJECT return a low pass filter mfir(). A Butterworth filter is used.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MKBANDREJECT return a low pass filter mfir().
0005 %               A Butterworth filter is used.
0006 %
0007 % CALL:        f = mkbandreject(f, pl)
0008 %
0009 % VERSION: $Id: mkbandreject.m,v 1.3 2008/08/06 15:03:49 ingo Exp $
0010 %
0011 % HISTORY: 27-08-2002 M Hewitson
0012 %           Creation
0013 %
0014 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0015 %
0016 function f = mkbandreject(f, pl)
0017
0018     g      = find(pl, 'gain');
0019     fc     = find(pl, 'fc');
0020     fs     = find(pl, 'fs');
0021     order  = find(pl, 'order');
0022     win    = find(pl, 'win');
0023
0024     if(fc > fs/2)
0025         error('fc must be < fs/2');
0026     end
0027     if(fc(1) > fc(2))
0028         error('fc(1) must be < fc(2)');
```

```
0029 end
0030
0031 f.name      = 'std bandreject';
0032 f.fs        = fs;
0033 f.a         = g.*fir1(order, 2.*fc/fs, 'stop', win.win);
0034 f.gd        = (f.ntaps)/2;
0035 f.histout   = zeros(1,f.ntaps-1); % initialise output history
0036
0037
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

mkhighpass

PURPOSE ^

MKHIGHPASS return a high pass filter mfir(). A Butterworth filter is used.

SYNOPSIS ^

function f = mkhighpass(f, pl)

DESCRIPTION ^

MKHIGHPASS return a high pass filter mfir(). A Butterworth filter is used.

DESCRIPTION: MKHIGHPASS return a high pass filter mfir().
A Butterworth filter is used.

CALL: f = mkhighpass(f, pl)

VERSION: \$Id: mkhighpass.m,v 1.3 2008/08/06 15:03:49 ingo Exp \$

HISTORY: 27-08-2002 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  fromStandard Construct an mfir from a standard types
-  mfir MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKHIGHPASS return a high pass filter mfir(). A Butterworth filter is used.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MKHIGHPASS return a high pass filter mfir().
0005 %               A Butterworth filter is used.
0006 %
0007 % CALL:        f = mkhighpass(f, pl)
0008 %
0009 % VERSION: $Id: mkhighpass.m,v 1.3 2008/08/06 15:03:49 ingo Exp $
0010 %
0011 % HISTORY: 27-08-2002 M Hewitson
0012 %           Creation
0013 %
0014 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0015 %
0016 function f = mkhighpass(f, pl)
0017 %
0018     g      = find(pl, 'gain');
0019     fc     = find(pl, 'fc');
0020     fs     = find(pl, 'fs');
0021     order  = find(pl, 'order');
0022     win    = find(pl, 'win');
0023 %
0024     if(fc(1) > fs/2)
0025         error('fc must be < fs/2');
0026     end
0027 %
0028 %
```

```
0029 f.name      = 'std highpass';  
0030 f.fs        = fs;  
0031 f.a         = g.*fir1(order, 2*fc/fs, 'high', win.win);  
0032 f.gd        = (f.ntaps)/2;  
0033 f.histout   = zeros(1,f.ntaps-1); % initialise output history  
0034  
0035  
0036
```


mklowpass

PURPOSE ^

MKLOWPASS return a low pass filter mfir().

SYNOPSIS ^

function f = mklowpass(f, pl)

DESCRIPTION ^

```
MKLOWPASS return a low pass filter mfir().
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MKLOWPASS return a low pass filter mfir().

CALL:      f = mklowpass(f, pl)

VERSION: $Id: mklowpass.m,v 1.3 2008/08/06 15:03:49 ingo Exp $

HISTORY: 27-08-2002 M Hewitson
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [fromStandard](#) Construct an mfir from a standard types
-  [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKLOWPASS return a low pass filter mfir().
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MKLOWPASS return a low pass filter mfir().
0005 %
0006 % CALL:      f = mklowpass(f, pl)
0007 %
0008 % VERSION: $Id: mklowpass.m,v 1.3 2008/08/06 15:03:49 ingo Exp $
0009 %
0010 % HISTORY: 27-08-2002 M Hewitson
0011 %          Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014 %
0015 function f = mklowpass(f, pl)
0016
0017     g      = find(pl, 'gain');
0018     fc      = find(pl, 'fc');
0019     fs      = find(pl, 'fs');
0020     order   = find(pl, 'order');
0021     win     = find(pl, 'win');
0022
0023     if(fc(1) > fs/2)
0024         error('fc must be < fs/2');
0025     end
0026
0027     f.name   = 'std lowpass';
0028     f.fs     = fs;
0029     f.a      = g.*fir1(order, 2*fc/fs, 'low', win.win);
```

```
0030     f.gd      = (f.ntaps)/2;  
0031     f.histout = zeros(1,f.ntaps-1);    % initialise output history  
0032  
0033
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

parseFilterParams

PURPOSE ^

PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard FIR filter.

SYNOPSIS ^

```
function plo = parseFilterParams(pl)
```

DESCRIPTION ^

```
PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard
FIR filter.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: PARSEFILTERPARAMS parses the input plist and returns a full
             plist for designing a standard FIR filter. Defaults are used
             for those parameters missing from the input plist.

CALL:       plo = parseFilterParams(pl)

INPUT:      'type' - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'.
             [default: 'lowpass']
             'gain' - gain of filter [default: 1.0]
             'fs'  - sample frequency to design for [default: 1 Hz]
             'order' - order of filter [default: 64]
             'fc'  - corner frequencies. This is a two element vector for
                     bandpass and bandreject filters. [default: 0.1 or [0.1 0.25] Hz]
             'Win' - a window object to use in the design. [default: Hamming]

VERSION:    $Id: parseFilterParams.m,v 1.4 2008/08/08 14:52:24 hewitson Exp $

HISTORY:    11-02-2008 M Hewitson
             Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a
0002 % standard FIR filter.
0003 %
0004 % DESCRIPTION: PARSEFILTERPARAMS parses the input plist and returns a full
0005 %               plist for designing a standard FIR filter. Defaults are used
0006 %               for those parameters missing from the input plist.
0007 %
0008 % CALL:       plo = parseFilterParams(pl)
0009 %
0010 % INPUT:      'type' - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'.
0011 %             [default: 'lowpass']
0012 %             'gain' - gain of filter [default: 1.0]
0013 %             'fs'  - sample frequency to design for [default: 1 Hz]
0014 %             'order' - order of filter [default: 64]
0015 %             'fc'  - corner frequencies. This is a two element vector for
0016 %                     bandpass and bandreject filters. [default: 0.1 or [0.1 0.25]
```

```

0017 %           'Win'    - a window object to use in the design. [default: Hamming]
0018 %
0019 % VERSION:      $Id: parseFilterParams.m,v 1.4 2008/08/08 14:52:24 hewitson Exp $
0020 %
0021 % HISTORY:      11-02-2008 M Hewitson
0022 %                Creation
0023 %
0024 %%%%%%%%%%%%%%%
0025
0026 function plo = parseFilterParams(pl)
0027
0028     plo = plist();
0029
0030     % type
0031     type = find(pl, 'type');
0032     if isempty(type)
0033         type = 'lowpass';
0034         utils.helper.msg(msg.OPROC2, 'using default type 'lowpass');
0035     end
0036     plo = append(plo, 'type', type);
0037
0038     % gain
0039     gain = find(pl, 'gain');
0040     if isempty(gain)
0041         gain = 1.0;
0042         utils.helper.msg(msg.OPROC2, ['using default gain ' num2str(gain)]);
0043     end
0044     plo = append(plo, 'gain', gain);
0045
0046     % order
0047     order = find(pl, 'order');
0048     if isempty(order)
0049         order = 64;
0050         utils.helper.msg(msg.OPROC2, ['- using default order ' num2str(order)]);
0051     end
0052     if mod(order,2) == 1
0053         warning('!!! resetting filter order to even number (+1)')
0054         order = order + 1;
0055     end
0056     plo = append(plo, 'order', order);
0057
0058     % fc
0059     fc = find(pl, 'fc');
0060     if isempty(fc)
0061         if strcmp(type, 'bandreject') || strcmp(type, 'bandpass')
0062             fc = [0.1 0.25];
0063         else
0064             fc = 0.1;
0065         end
0066         utils.helper.msg(msg.OPROC2, ['- using default fc ' num2str(fc)]);
0067     end
0068     plo = append(plo, 'fc', fc);
0069
0070     % fs
0071     fs = find(pl, 'fs');
0072     if isempty(fs)
0073         fs = 10*max(fc);
0074         warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2fHz.', fs)...
0075             sprintf('\nThe filter will be redesigned later when used.')] );
0076     end
0077     plo = append(plo, 'fs', fs);
0078
0079     % win
0080     win = find(pl, 'Win');
0081     if isempty(win)
0082         % then we use the default window
0083         win = specwin('Hamming', order+1);
0084     end
0085     if length(win.win) ~= order + 1
0086         warning('!!! setting window length to filter order !!!');
0087         switch win.type
0088             case {'Kaiser', 'Flattop'}
0089                 win = specwin(win.type, order + 1, win.psl1);
0090             otherwise
0091                 win = specwin(win.type, order + 1);
0092         end
0093     end
0094     plo = append(plo, 'Win', win);
0095 end
0096
0097

```

setGd

PURPOSE ^

SETGD Set the property 'gd'

SYNOPSIS ^

function obj = setGd(obj, val)

DESCRIPTION ^

```
SETGD Set the property 'gd'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETGD Set the property 'gd'

CALL:      obj = obj.setGd(12);
           obj = setGd(obj, 12);

INPUTS:    obj - is a mfir object

VERSION:    $Id: setGd.m,v 1.1 2008/07/22 10:22:38 ingo Exp $

HISTORY:    16-07-2008 Diepholz
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

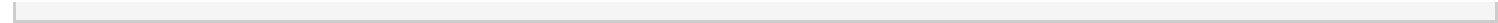
 [copy](#) COPY Make copy of mfir objects depending of the second input

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % SETGD Set the property 'gd'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETGD Set the property 'gd'
0005 %
0006 % CALL:      obj = obj.setGd(12);
0007 %           obj = setGd(obj, 12);
0008 %
0009 % INPUTS:    obj - is a mfir object
0010 %
0011 % VERSION:    $Id: setGd.m,v 1.1 2008/07/22 10:22:38 ingo Exp $
0012 %
0013 % HISTORY:    16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function obj = setGd(obj, val)
0019     %% decide whether we modify the mfir-object, or create a new one.
0020     obj = copy(obj, nargout);
0021 %
0022     %% set 'gd'
0023     obj.gd = val;
0024 end
0025
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [mfir](#) MFIR FIR filter object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034         if ~isempty(obj_struct.gain) && obj_struct.gain ~= 0
0035             obj_struct.a = obj_struct.a * obj_struct.gain;
0036             obj_struct = rmfield(obj_struct, 'gain');
0037         end
0038         if isempty(obj_struct.plist)
0039             obj_struct.plist = plist();
0040         end
0041         ii = minfo('update_struct', 'mfir', '', '', ...
0042             '$Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $', ...
0043             {'Default'}, plist(), 0, 0);
0044         obj_struct.hist = history(time().utc_epoch_milli, ii, obj_struct.plist);
0045         obj_struct = rmfield(obj_struct, 'plist');
0046         obj_struct.creator = provenance('Updater 1.0 -> 1.1');
0047         obj_struct.gd = 0;
0048     catch ME
0049         disp(varargin{1});
0050         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (mfir) above is not
0051 from version 1.0')));
0052     end
0053     % Set the struct_ver to '1.1'
0054     struct_ver = '1.1';
0055 end
0056
0057 %%%%%%%%%% Update version '1.1 -> 1.2' %%%%%%%%%%
0058
0059 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0060     strcmp(struct_ver, '1.1')
0061     % Set the struct_ver to '1.2'
0062     struct_ver = '1.1';
0063 end
0064
0065 %%%%%%%%%% Update version '1.2 -> 1.3' %%%%%%%%%%
0066
0067 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0068     strcmp(struct_ver, '1.2')
0069     % Set the struct_ver to '1.3'
0070     struct_ver = '1.3';
0071 end
0072
0073 varargout{1} = obj_struct;
0074
0075

```


ltpda_filter

PURPOSE ^

LTPDA_FILTER is the abstract base class for ltpda filter objects.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_FILTER is the abstract base class for ltpda filter objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_FILTER is the abstract base class for ltpda filter objects.

SUPER CLASSES: ltpda_uoh < ltpda_uo < ltpda_obj

SUB CLASSES:   miir, mfir

PROPERTIES:

    Inherit Properties (read only)
    name          - name of object
    created       - creation time (time-object)
    prov         - contains a instance of the provenance class.
    hist         - history of the object (history object)
    version       - cvs-version string.

    Protected Properties (read only)
    fs           - sample rate that the filter is designed for
    infile       - filename if the filter was loaded from file
    a            - set of numerator coefficients
    histout      - input history values to filter

LTPDA_FILTER Methods:

    Public Methods
    setHistout - set the property 'histout'

    Protected Methods
    setA       - set the property 'a'
    setFs      - set the property 'fs'
    setInfile  - set the property 'infile'

M-FILE INFO: The following call returns an minfo object that contains
              information about the ltpda_filter constructor:
              >> info = ltpda_filter.getInfo
              or  >> info = ltpda_filter.getInfo('ltpda_filter')

VERSION:      $Id: ltpda_filter.m,v 1.14 2008/09/03 16:32:47 hewitson Exp $

HISTORY:      09-06-2008 Hewitson
               Creation.

SEE ALSO:     miir, mfir, ltpda_uoh, ltpda_uo, ltpda_obj
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpda_filter](#) LTPDA_FILTER is the abstract base class for ltpda filter objects.
-  [setA](#) SETA Set the property 'a'
-  [setFs](#) SETFS Set the property 'fs'
-  [setHistout](#) SETHISTOUT Set the property 'histout'

 **setInfile** SETINFILE Set the property 'infile'

This function is called by:

 **ltpda_filter** LTPDA_FILTER is the abstract base class for ltpda filter objects.

SUBFUNCTIONS ^

-  function obj = set.fs(obj, val)
-  function obj = set.infile(obj, val)
-  function obj = set.histout(obj, val)
-  function obj = set.a(obj, val)
-  function obj = ltpda_filter(varargin)
-  function ii = getInfo(varargin)
-  function out = VEROUT()
-  function out = SETS()
-  function out = getDefaultPlist()

SOURCE CODE ^

```
0001 % LTPDA_FILTER is the abstract base class for ltpda filter objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LTPDA_FILTER is the abstract base class for ltpda filter objects.
0005 %
0006 % SUPER CLASSES: ltpda_uoh < ltpda_uo < ltpda_obj
0007 %
0008 % SUB CLASSES:   miir, mfir
0009 %
0010 % PROPERTIES:
0011 %
0012 %     Inherit Properties (read only)
0013 %         name      - name of object
0014 %         created   - creation time (time-object)
0015 %         prov      - contains a instance of the provenance class.
0016 %         hist      - history of the object (history object)
0017 %         version   - cvs-version string.
0018 %
0019 %     Protected Properties (read only)
0020 %         fs         - sample rate that the filter is designed for
0021 %         infile     - filename if the filter was loaded from file
0022 %         a          - set of numerator coefficients
0023 %         histout    - input history values to filter
0024 %
0025 % LTPDA_FILTER Methods:
0026 %
0027 %     Public Methods
0028 %         setHistout - set the property 'histout'
0029 %
0030 %     Protected Methods
0031 %         setA       - set the property 'a'
0032 %         setFs      - set the property 'fs'
0033 %         setInfile  - set the property 'infile'
0034 %
0035 % M-FILE INFO: The following call returns an minfo object that contains
0036 %               information about the ltpda_filter constructor:
0037 %               >> info = ltpda_filter.getInfo
0038 %               or   >> info = ltpda_filter.getInfo('ltpda_filter')
0039 %
0040 % VERSION:      $Id: ltpda_filter.m,v 1.14 2008/09/03 16:32:47 hewitson Exp $
0041 %
0042 % HISTORY:      09-06-2008 Hewitson
0043 %               Creation.
0044 %
0045 % SEE ALSO:     miir, mfir, ltpda_uoh, ltpda_uo, ltpda_obj
0046 %
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048 %
0049 classdef ltpda_filter < ltpda_uoh
0050 %
0051 %
0052 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0053 %               Property definition
```

```

0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 %----- Public (read/write) Properties -----
0057 properties
0058 end
0059
0060 %----- Protected read-only Properties -----
0061 properties (SetAccess = protected)
0062     fs      = [];
0063     infile  = '';
0064     a       = [];
0065     histout = [];
0066 end
0067
0068 %----- Private Properties -----
0069 properties (GetAccess = protected, SetAccess = protected)
0070 end
0071
0072 %----- Abstract Properties -----
0073 properties (Abstract = true, SetAccess = protected)
0074 end
0075
0076 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077 %                               Check property setting                               %
0078 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0079
0080 methods
0081     function obj = set.fs(obj, val)
0082         if ~isempty(val)
0083             if ~isnumeric(val) || ~isreal(val) || val < 0
0084                 error('### The value for the property ''fs'' must be a real positive
number');
0085             end
0086             obj.fs = val;
0087         end
0088     end
0089     function obj = set.infile(obj, val)
0090         if ~(ischar(val) || isempty(val))
0091             error('### The value for the property ''infile'' must be a string');
0092         end
0093         obj.infile = val;
0094     end
0095     function obj = set.histout(obj, val)
0096         if ~isempty(val)
0097             if ~isnumeric(val) || ~isreal(val)
0098                 error('### The value for the property ''histout'' must be a real number(s)');
0099             end
0100         end
0101         obj.histout = val;
0102     end
0103     function obj = set.a(obj, val)
0104         if ~isempty(val)
0105             if ~isnumeric(val) || ~isreal(val)
0106                 error('### The value for the property ''a'' must be a real number(s)');
0107             end
0108         end
0109         obj.a = val;
0110     end
0111 end
0112
0113 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0114 %                               Constructor                               %
0115 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0116
0117 methods
0118     function obj = ltpda_filter(varargin)
0119
0120         %%% Call superclass
0121         obj = obj@ltpda_uoh(varargin{:});
0122
0123         %%% Set default values
0124
0125         if nargin == 1
0126             %%% obj = ltpda_uoh(struct)
0127             if isstruct(varargin{1})
0128                 %%% Set properties which are declared in this class
0129                 filter_struct = varargin{1};
0130                 obj.fs      = filter_struct.fs;
0131                 obj.infile  = filter_struct.infile;
0132                 obj.a       = filter_struct.a;
0133                 obj.histout = filter_struct.histout;
0134             end
0135         end
0136
0137     end
0138 end

```

```
0139
0140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141 %                                     Methods (public)                                     %
0142 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0143
0144 methods
0145     varargout = setHistout(varargin)
0146 end
0147
0148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0149 %                                     Methods (protected)                               %
0150 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0151
0152 methods (Access = protected)
0153     obj = setA(obj, val)
0154     obj = setFs(obj, val)
0155     obj = setInfile(obj, val)
0156 end
0157
0158 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0159 %                                     Methods (static)                                   %
0160 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0161
0162 methods (Static)
0163     function ii = getInfo(varargin)
0164         ii = utils.helper.generic_getInfo(varargin{:}, 'ltpda_filter');
0165     end
0166
0167     function out = VEROUT()
0168         out = '$Id: ltpda_filter.m,v 1.14 2008/09/03 16:32:47 hewitson Exp $';
0169     end
0170
0171     function out = SETS()
0172         out = {};
0173     end
0174
0175     function out = getDefaultPlist()
0176         out = [];
0177     end
0178
0179 end
0180
0181 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0182 %                                     Methods (abstract)                               %
0183 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0184
0185 methods (Abstract)
0186 end
0187
0188 end
0189
```

[Home](#) > [classes](#) > [@miir](#) > filload.m

filload

PURPOSE ^

Load a LISO *_iir.fil file to get the filter taps and return a

SYNOPSIS ^

function f = filload(filename)

DESCRIPTION ^

Load a LISO *_iir.fil file to get the filter taps and return a miir type object.

M Hewitson 11-02-07

\$Id: filload.m,v 1.4 2008/08/08 12:12:04 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 function f = filload(filename)
0002
0003 % Load a LISO *_iir.fil file to get the filter taps and return a
0004 % miir type object.
0005 %
0006 % M Hewitson 11-02-07
0007 %
0008 % $Id: filload.m,v 1.4 2008/08/08 12:12:04 hewitson Exp $
0009 %
0010
0011 delims = ' \n';
0012 in = fopen(filename, 'rt');
0013
0014 name = filename(1:end-4);
0015
0016 if(in < 0)
0017     error(['can not open file ' filename]);
0018 end
0019 lbuffer = fgets(in);
0020 n = 1;
0021 ncoeffs = 0;
0022 nbcoeffs = 0;
0023 filt.a = [];
0024 filt.b = [];
0025 norminst = '';
0026
0027 while (lbuffer > 0)
0028     if(length(lbuffer) > 1)
0029         % read the first token on this line
0030         [token, r] = strtok (lbuffer, delims);
0031
0032         % Get fs
0033         if(strcmp(token, 'iir'))
0034             [token, r] = strtok (r, delims); % value
0035             fs = str2double(token);
0036         end
0037
0038         % Get a coeffs
```

```

0039     if(token(1) == 'a')
0040         [token, r] = strtok (r, delims); % value
0041         nacoefss = nacoefss+1;
0042         filt.a(nacoefss) = str2double(token);
0043     end
0044     % Get b coefss
0045     if(token(1) == 'b')
0046         [token, r] = strtok (r, delims); % value
0047         nbcoefss = nbcoefss+1;
0048         filt.b(nbcoefss) = str2double(token);
0049     end
0050 end
0051
0052     n = n + 1;
0053     lbuffer = fgets(in);
0054 end
0055 fclose(in);
0056
0057 if(nacoefss > nbcoefss)
0058     error('## unstable filter: nacoefss > nbcoefss');
0059 end
0060
0061 if(nacoefss < nbcoefss)
0062     filt.a = [filt.a zeros(1, (nbcoefss-nacoefss))];
0063 end
0064
0065 ncoefss = nbcoefss;
0066
0067 f.name      = name;
0068 f.fs        = fs;
0069 f.a         = filt.a;
0070 f.b         = filt.b;
0071 f.ntaps     = ncoefss;
0072 f.gain      = 1;
0073 f.histin    = zeros(1, f.ntaps-1);
0074 f.histout   = zeros(1, f.ntaps-1);
0075 end
0076
0077

```

fromAB

PURPOSE ^

Construct an miir from coefficients

SYNOPSIS ^

function f = fromAB(f, pli)

DESCRIPTION ^

Construct an miir from coefficients
%%
FUNCTION: fromAB
DESCRIPTION: Construct an miir from coefficients
CALL: f = fromAB(f, pli)
PARAMETER: pli: Parameter list object
HISTORY: 09-06-2008 M Hewitson
Creation
%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 **miir** MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % Construct an miir from coefficients
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION: fromAB
0005 %
0006 % DESCRIPTION: Construct an miir from coefficients
0007 %
0008 % CALL: f = fromAB(f, pli)
0009 %
0010 % PARAMETER: pli: Parameter list object
0011 %
0012 % HISTORY: 09-06-2008 M Hewitson
0013 % Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 function f = fromAB(f, pli)
0018
0019 ii = miir.getInfo('miir', 'From AB');
0020 % Set the method version string in the minfo object
0021 ii.setMversion(['$Id: fromAB.m,v 1.9 2008/09/04 13:37:14 ingo Exp $-->' ii.mversion]);
0022
0023 % Add default values
0024 pl = combine(pli, ii.plist);
0025
0026 a = find(pl, 'a');
0027 b = find(pl, 'b');
0028 fs = find(pl, 'fs');
0029
```

```
0030 % Checking the coefficients are listed in rows
0031 if size(a,1)~=1
0032     a = a';
0033 end
0034 if size(b,1)~=1
0035     b = b';
0036 end
0037
0038 % Zero pad to avoid 0 length history vector
0039 if numel(a) <= 1
0040     a = [a 0];
0041 end
0042 if numel(b) <= 1
0043     b = [b 0];
0044 end
0045
0046 f.setName('AB', 'internal');
0047 f.setFs(fs);
0048 f.setA(a);
0049 f.setB(b);
0050 f.setHistin(zeros(1,f.ntaps-1)); % initialise input history
0051 f.setHistout(zeros(1,f.ntaps-1)); % initialise output history
0052
0053 % Add history
0054 f.addHistory(ii, pl, [], []);
0055
0056 end % End fromAB
```


fromLISO

PURPOSE ^

FROMLISO Construct a miir filter from a LISO file

SYNOPSIS ^

function f = fromLISO(f, pli)

DESCRIPTION ^

```
FROMLISO Construct a miir filter from a LISO file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromLISO

DESCRIPTION: Construct a miir filter from a LISO file

CALL:         f = fromLISO(f, pli)

PARAMETER:    f:   Empty miir-object
               pli: input plist (must contain the filename)

VERSION:      $Id: fromLISO.m,v 1.1 2008/08/22 14:17:26 ingo Exp $

HISTORY:      22-03-2008 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 **miir** MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % FROMLISO Construct a miir filter from a LISO file
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromLISO
0005 %
0006 % DESCRIPTION: Construct a miir filter from a LISO file
0007 %
0008 % CALL:         f = fromLISO(f, pli)
0009 %
0010 % PARAMETER:    f:   Empty miir-object
0011 %               pli: input plist (must contain the filename)
0012 %
0013 % VERSION:      $Id: fromLISO.m,v 1.1 2008/08/22 14:17:26 ingo Exp $
0014 %
0015 % HISTORY:      22-03-2008 M Hewitson
0016 %               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 function f = fromLISO(f, pli)
0021
0022     VERSION = '$Id: fromLISO.m,v 1.1 2008/08/22 14:17:26 ingo Exp $';
0023
0024     ii = f.getInfo(class(f), 'From LISO File');
0025
0026     % Set the method version string in the minfo object
```

```
0027 ii.setMversion([VERSION '-->' ii.mversion]);
0028
0029 filename = find(pli, 'filename');
0030 filt     = miir.filload(filename);
0031
0032 f.name    = filt.name;
0033 f.fs      = filt.fs;
0034 f.a       = filt.gain*filt.a;
0035 f.b       = filt.b;
0036 f.histin  = filt.histin;
0037 f.histout = filt.histout;
0038 f.infile  = filename;
0039
0040 f.addHistory(ii, pli, [], []);
0041
0042 end % pzm = pzmFromLISO(filename, version, algoname)
0043
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromPzmodel

PURPOSE ^

Construct an miir from a pzmodel

SYNOPSIS ^

function f = fromPzmodel(f, pli)

DESCRIPTION ^

```
Construct an miir from a pzmodel
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromPzmodel

DESCRIPTION: Construct an miir from a pzmodel

CALL:         f = fromPzmodel(f, pli)

PARAMETER:    type:      String with filter type description
               pli:      Parameter list object

HISTORY:      09-06-2008 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % Construct an miir from a pzmodel
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromPzmodel
0005 %
0006 % DESCRIPTION: Construct an miir from a pzmodel
0007 %
0008 % CALL:         f = fromPzmodel(f, pli)
0009 %
0010 % PARAMETER:    type:      String with filter type description
0011 %               pli:      Parameter list object
0012 %
0013 % HISTORY:      09-06-2008 M Hewitson
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function f = fromPzmodel(f, pli)
0018
0019     VERSION = '$Id: fromPzmodel.m,v 1.7 2008/09/04 13:37:14 ingo Exp $';
0020     ii = miir.getInfo('miir', 'From Pzmodel');
0021     % Set the method version string in the minfo object
0022     ii.setMversion([VERSION '-->' ii.mversion]);
0023
0024     % Add default values
0025     pl = combine(pli, ii.plist);
0026
0027     % Get parameters
0028     pzm = find(pl, 'pzmodel');
```

```
0029 fs = find(pl, 'fs');
0030
0031 if isempty(fs)
0032     % get max freq in pzmodel
0033     fs = 8*getupperFreq(pzm);
0034     warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2f Hz.', fs)...
0035         sprintf('\nThe filter will be redesigned later when used.')] );
0036 end
0037 % make MIIR filter
0038 f = tomiir(pzm, fs);
0039 f.setName(pzm.name, 'internal');
0040
0041 % Add history
0042 f.addHistory(ii, pl, [], pzm.hist);
0043
0044 end % End fromPzmodel
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromStandard

PURPOSE ^

Construct an miir from a standard types

SYNOPSIS ^

function f = fromStandard(f, pli)

DESCRIPTION ^

```
Construct an miir from a standard types
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromStandard

DESCRIPTION: Construct an miir from a standard types

CALL:         f = fromStandard(f, pli)

PARAMETER:    type:      String with filter type description
               pli:      Parameter list object

HISTORY:      22-03-2008 M Hueller
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [mkbandpass](#) MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
-  [mkbandreject](#) MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.
-  [mkhighpass](#) MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.
-  [mklowpass](#) MKLOWPASS return a low pass filter miir(). A Butterworth filter is used.

This function is called by:

-  [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % Construct an miir from a standard types
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromStandard
0005 %
0006 % DESCRIPTION: Construct an miir from a standard types
0007 %
0008 % CALL:         f = fromStandard(f, pli)
0009 %
0010 % PARAMETER:    type:      String with filter type description
0011 %               pli:      Parameter list object
0012 %
0013 % HISTORY:      22-03-2008 M Hueller
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function f = fromStandard(f, pli)
0018
0019     VERSION = '$Id: fromStandard.m,v 1.5 2008/09/04 13:37:14 ingo Exp $';
0020
```

```
0021 ii = miir.getInfo('miir', 'From Standard Type');
0022 % Set the method version string in the minfo object
0023 ii.setMversion([VERSION '-->' ii.mversion]);
0024
0025 % Add default values
0026 pl = combine(pli, ii.plist);
0027
0028 % Get parameters
0029 type = find(pl, 'type');
0030
0031 % check and fill parameter list
0032 plo = miir.parseFilterParams(pl);
0033 switch type
0034     case 'lowpass'
0035         f = mklowpass(f, plo);
0036     case 'highpass'
0037         f = mkhighpass(f, plo);
0038     case 'bandpass'
0039         f = mkbandpass(f, plo);
0040     case 'bandreject'
0041         f = mkbandreject(f, plo);
0042     otherwise
0043         error('### unknown standard filter type in miir constructor.');
```

```
0044 end
0045
0046 % Add history
0047 f.addHistory(ii, pl, [], []);
0048
0049 end % function f = miirFromStandardType(type, pli, version, algoname)
```

mkbandpass

PURPOSE ^

MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.

SYNOPSIS ^

function f = mkbandpass(f, pl)

DESCRIPTION ^

MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
DESCRIPTION: MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
CALL: f = mkbandpass(f, pl)
VERSION: \$Id: mkbandpass.m,v 1.4 2008/08/06 15:03:50 ingo Exp \$
HISTORY: 27-08-2002 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [fromStandard](#) Construct an miir from a standard types
-  [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MKBANDPASS return a bandpass filter miir(). A Cheby filter is used.
0005 %
0006 % CALL: f = mkbandpass(f, pl)
0007 %
0008 % VERSION: $Id: mkbandpass.m,v 1.4 2008/08/06 15:03:50 ingo Exp $
0009 %
0010 % HISTORY: 27-08-2002 M Hewitson
0011 % Creation
0012 %
0013 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0014 %
0015 function f = mkbandpass(f, pl)
0016
0017 g = find(pl, 'gain');
0018 fc = find(pl, 'fc');
0019 fs = find(pl, 'fs');
0020 order = find(pl, 'order');
0021 ripple = find(pl, 'ripple');
0022
0023 if(fc > fs/2)
0024     error('fc must be < fs/2');
0025 end
0026 if(fc(1) > fc(2))
0027     error('fc(1) must be < fc(2)');
0028 end
0029
```

```
0030 % Build filter coefficients
0031 [a, b] = cheby1(order, ripple, 2.*fc./fs);
0032
0033 % Set filter properties
0034 f.setName('bandpass', 'internal');
0035 f.setFs(fs);
0036 f.setA(g.*a);
0037 f.setB(b);
0038 f.setHistin(zeros(1,f.ntaps-1)); % initialise input history
0039 f.setHistout(zeros(1,f.ntaps-1)); % initialise output history
0040
0041
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

mkbandreject

PURPOSE ^

MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.

SYNOPSIS ^

function f = mkbandreject(f, pl)

DESCRIPTION ^

MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.
DESCRIPTION: MKBANDREJECT return a low pass filter miir().
A Butterworth filter is used.
CALL: f = mkbandreject(f, pl)
VERSION: \$Id: mkbandreject.m,v 1.4 2008/08/06 15:03:50 ingo Exp \$
HISTORY: 27-08-2002 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- fromStandard Construct an miir from a standard types
- miir MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKBANDREJECT return a low pass filter miir(). A Butterworth filter is used.
0002 %
0003 %
0004 % DESCRIPTION: MKBANDREJECT return a low pass filter miir().
0005 % A Butterworth filter is used.
0006 %
0007 % CALL: f = mkbandreject(f, pl)
0008 %
0009 % VERSION: $Id: mkbandreject.m,v 1.4 2008/08/06 15:03:50 ingo Exp $
0010 %
0011 % HISTORY: 27-08-2002 M Hewitson
0012 % Creation
0013 %
0014 %
0015
0016 function f = mkbandreject(f, pl)
0017
0018 g = find(pl, 'gain');
0019 fc = find(pl, 'fc');
0020 fs = find(pl, 'fs');
0021 order = find(pl, 'order');
0022 ripple = find(pl, 'ripple');
0023
0024 if(fc > fs/2)
0025     error('fc must be < fs/2');
0026 end
0027 if(fc(1) > fc(2))
0028     error('fc(1) must be < fc(2)');
```

```
0029 end
0030
0031 % Build filter coefficients
0032 [a, b] = cheby1(order, ripple, 2.*fc./fs, 'stop');
0033
0034 % Set filter properties
0035 f.setName('bandreject', 'internal');
0036 f.setFs(fs);
0037 f.setA(g.*a);
0038 f.setB(b);
0039 f.setHistin(zeros(1, f.ntaps-1)); % initialise input history
0040 f.setHistout(zeros(1, f.ntaps-1)); % initialise output history
0041
0042
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

mkhighpass

PURPOSE ^

MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.

SYNOPSIS ^

function f = mkhighpass(f, pl)

DESCRIPTION ^

```
MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: MKHIGHPASS return a high pass filter miir().
             A Butterworth filter is used.

CALL:       f = mkhighpass(f, pl)

VERSION: $Id: mkhighpass.m,v 1.4 2008/08/06 15:03:50 ingo Exp $

HISTORY: 27-08-2002 M Hewitson
          Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [fromStandard](#) Construct an miir from a standard types
-  [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKHIGHPASS return a high pass filter miir(). A Butterworth filter is used.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MKHIGHPASS return a high pass filter miir().
0005 %             A Butterworth filter is used.
0006 %
0007 % CALL:       f = mkhighpass(f, pl)
0008 %
0009 % VERSION: $Id: mkhighpass.m,v 1.4 2008/08/06 15:03:50 ingo Exp $
0010 %
0011 % HISTORY: 27-08-2002 M Hewitson
0012 %             Creation
0013 %
0014 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0015 %
0016 function f = mkhighpass(f, pl)
0017 %
0018     g      = find(pl, 'gain');
0019     fc      = find(pl, 'fc');
0020     fs      = find(pl, 'fs');
0021     order   = find(pl, 'order');
0022 %
0023 %
0024     if(fc > fs/2)
0025         error('fc must be < fs/2');
0026     end
0027 %
0028 % Build filter coefficients
```

```
0029 [a, b] = butter(order, 2*fc/fs, 'high');
0030
0031 % Set filter properties
0032 f.setName('highpass', 'internal');
0033 f.setFs(fs);
0034 f.setA(g.*a);
0035 f.setB(b);
0036 f.setHistin(zeros(1, f.ntaps-1)); % initialise input history
0037 f.setHistout(zeros(1, f.ntaps-1)); % initialise output history
0038
0039
```

mklowpass

PURPOSE ^

MKLOWPASS return a low pass filter miir(). A Butterworth filter is used.

SYNOPSIS ^

function f = mklowpass(f, pl)

DESCRIPTION ^

MKLOWPASS return a low pass filter miir(). A Butterworth filter is used.
DESCRIPTION: MKLOWPASS return a low pass filter miir().
A Butterworth filter is used.
CALL: f = mklowpass(f, pl)
VERSION: \$Id: mklowpass.m,v 1.4 2008/08/06 15:03:50 ingo Exp \$
HISTORY: 27-08-2002 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  fromStandard Construct an miir from a standard types
-  miir MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % MKLOWPASS return a low pass filter miir(). A Butterworth filter is used.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: MKLOWPASS return a low pass filter miir().
0005 %               A Butterworth filter is used.
0006 %
0007 % CALL:        f = mklowpass(f, pl)
0008 %
0009 % VERSION: $Id: mklowpass.m,v 1.4 2008/08/06 15:03:50 ingo Exp $
0010 %
0011 % HISTORY: 27-08-2002 M Hewitson
0012 %           Creation
0013 %
0014 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0015 %
0016 function f = mklowpass(f, pl)
0017
0018     g      = find(pl, 'gain');
0019     fc      = find(pl, 'fc');
0020     fs      = find(pl, 'fs');
0021     order   = find(pl, 'order');
0022
0023     if(fc > fs/2)
0024         error('fc must be < fs/2');
0025     end
0026
0027     % Build coefficients
0028     [a, b] = butter(order, 2*fc/fs);
```

```
0029
0030 % Set filter properties
0031 f.setName('lowpass', 'internal');
0032 f.setFs(fs);
0033 f.setA(a.*g);
0034 f.setB(b);
0035 f.setHistin(zeros(1,f.ntaps-1)); % initialise input history
0036 f.setHistout(zeros(1,f.ntaps-1)); % initialise output history
0037
0038
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

parseFilterParams

PURPOSE ^

PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard IIR filter.

SYNOPSIS ^

function plo = parseFilterParams(pl)

DESCRIPTION ^

PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard IIR filter.

DESCRIPTION: PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a standard IIR filter. Defaults are used for those parameters missing from the input plist.

CALL: plo = parseFilterParams(pl)

INPUT: 'type' - one of 'highpass', 'lowpass', 'bandpass', 'bandreject' [default: 'lowpass']
'gain' - gain of filter [default: 1.0]
'fs' - sample frequency to design for [default: 1 Hz]
'order' - order of filter [default: 1]
'fc' - corner frequencies. This is a two element vector for bandpass and bandreject filters. [default: 0.1 or [0.1 0.25] Hz]

VERSION: \$Id: parseFilterParams.m,v 1.4 2008/08/08 14:52:24 hewitson Exp \$

HISTORY: 11-02-2008 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % PARSEFILTERPARAMS parses the input plist and returns a full plist for designing a
0002 % standard IIR filter.
0003 %
0004 % DESCRIPTION: PARSEFILTERPARAMS parses the input plist and returns a
0005 % full plist for designing a standard IIR filter. Defaults are used
0006 % for those parameters missing from the input plist.
0007 %
0008 % CALL: plo = parseFilterParams(pl)
0009 %
0010 % INPUT: 'type' - one of 'highpass', 'lowpass', 'bandpass', 'bandreject'
0011 % [default: 'lowpass']
0012 % 'gain' - gain of filter
0013 % [default: 1.0]
```

```

0014 % 'fs' - sample frequency to design for
0015 % [default: 1 Hz]
0016 % 'order' - order of filter
0017 % [default: 1]
0018 % 'fc' - corner frequencies. This is a two element vector for
0019 % bandpass and bandreject filters.
0020 % [default: 0.1 or [0.1 0.25] Hz]
0021 %
0022 % VERSION: $Id: parseFilterParams.m,v 1.4 2008/08/08 14:52:24 hewitson Exp $
0023 %
0024 % HISTORY: 11-02-2008 M Hewitson
0025 % Creation
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 function plo = parseFilterParams(pl)
0030
0031 plo = plist();
0032
0033 % type
0034 type = find(pl, 'type');
0035 if isempty(type)
0036 type = 'lowpass';
0037 utils.helper.msg(msg.OPROC2, 'using default type 'lowpass'');
0038 end
0039 plo = append(plo, param('type', type));
0040
0041 % gain
0042 gain = find(pl, 'gain');
0043 if isempty(gain)
0044 gain = 1.0;
0045 utils.helper.msg(msg.OPROC2, ['using default gain ' num2str(gain)]);
0046 end
0047 plo = append(plo, param('gain', gain));
0048
0049 % order
0050 order = find(pl, 'order');
0051 if isempty(order)
0052 order = 1.0;
0053 utils.helper.msg(msg.OPROC2, ['using default order ' num2str(order)]);
0054 end
0055 plo = append(plo, param('order', order));
0056
0057 % fc
0058 fc = find(pl, 'fc');
0059 if isempty(fc)
0060 if strcmp(type, 'bandreject') || strcmp(type, 'bandpass')
0061 fc = [0.1 0.25];
0062 else
0063 fc = 0.1;
0064 end
0065 utils.helper.msg(msg.OPROC2, ['using default fc ' num2str(fc)]);
0066 end
0067 plo = append(plo, param('fc', fc));
0068
0069 % fs
0070 fs = find(pl, 'fs');
0071 if isempty(fs)
0072 fs = 10*max(fc);
0073 warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2fHz.', fs)...
0074 sprintf('\nThe filter will be redesigned later when used.')] );
0075 end
0076 % Increase fs until the cutoff is ok
0077 while fs < 2*fc
0078 fs = fs*2;
0079 end
0080 plo = append(plo, param('fs', fs));
0081
0082 % ripple
0083 ripple = find(pl, 'ripple');
0084 if isempty(ripple)
0085 ripple = 0.5;
0086 utils.helper.msg(msg.OPROC2, ['using default ripple ' num2str(ripple)]);
0087 end
0088 plo = append(plo, param('ripple', ripple));
0089 end
0090
0091
0092

```


setB

PURPOSE ^

SETB Set the property 'b'

SYNOPSIS ^

function obj = setB(obj, val)

DESCRIPTION ^

SETB Set the property 'b'

DESCRIPTION: SETB Set the property 'b'

CALL: obj = obj.setB(12);
obj = setB(obj, 12);

INPUTS: obj - is a miir object

VERSION: \$Id: setB.m,v 1.2 2008/07/22 10:22:39 ingo Exp \$

HISTORY: 16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

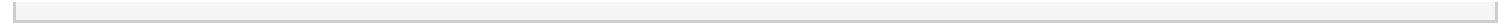
 [copy](#) COPY Make copy of miir objects depending of the second input

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % SETB Set the property 'b'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETB Set the property 'b'
0005 %
0006 % CALL:      obj = obj.setB(12);
0007 %           obj = setB(obj, 12);
0008 %
0009 % INPUTS:    obj - is a miir object
0010 %
0011 % VERSION:   $Id: setB.m,v 1.2 2008/07/22 10:22:39 ingo Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function obj = setB(obj, val)
0019 %%% decide whether we modify the miir-object, or create a new one.
0020 obj = copy(obj, nargout);
0021 %
0022 %%% set 'b'
0023 obj.b = val;
0024 end
0025
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [miir](#) MIIR IIR filter object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034         if ~isempty(obj_struct.gain) && obj_struct.gain ~= 0
0035             obj_struct.a = obj_struct.a * obj_struct.gain;
0036             obj_struct = rmfield(obj_struct, 'gain');
0037         end
0038         if isempty(obj_struct.plist)
0039             obj_struct.plist = plist();
0040         end
0041         ii = minfo('update_struct', 'miir', '', '', ...
0042             '$Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $', ...
0043             {'Default'}, plist(), 0, 0);
0044         obj_struct.hist = history(time().utc_epoch_milli, ii, obj_struct.plist);
0045         obj_struct = rmfield(obj_struct, 'plist');
0046         obj_struct.creator = provenance('Updater 1.0 -> 1.1');
0047     catch ME
0048         disp(varargin{1});
0049         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (miir) above is not
0050 from version 1.0')));
0051     end
0052     % Set the struct_ver to '1.1'
0053     struct_ver = '1.1';
0054 end
0055
0056 %%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0059     strcmp(struct_ver, '1.1')
0060     % Set the struct_ver to '1.2'
0061     struct_ver = '1.1';
0062 end
0063
0064 %%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065
0066 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0067     strcmp(struct_ver, '1.2')
0068     % Set the struct_ver to '1.3'
0069     struct_ver = '1.3';
0070 end
0071
0072 varargout{1} = obj_struct;
0073

```

computeInterval

PURPOSE ^

COMPUTEINTERVAL compute the interval of the time span.

SYNOPSIS ^

function ts = computeInterval(ts)

DESCRIPTION ^

```
COMPUTEINTERVAL compute the interval of the time span.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: COMPUTE_INTERVAL compute the interval of the time span.

CALL:      t1 = compute_interval(t1)

VERSION:    $Id: computeInterval.m,v 1.1 2008/07/23 17:31:56 ingo Exp $

HISTORY:    03-08-2007 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a timespan object into a string.

This function is called by:

-  [fromTimespanDef](#)
%%
-  [timespan](#) TIMESPAN timespan object class constructor.

SOURCE CODE ^

```
0001 % COMPUTEINTERVAL compute the interval of the time span.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COMPUTE_INTERVAL compute the interval of the time span.
0005 %
0006 % CALL:      t1 = compute_interval(t1)
0007 %
```

```

0008 % VERSION:      $Id: computeInterval.m,v 1.1 2008/07/23 17:31:56 ingo Exp $
0009 %
0010 % HISTORY:      03-08-2007 Diepholz
0011 %              Creation
0012 %
0013 %%%%%%%%%%%
0014
0015 function ts = computeInterval(ts)
0016
0017     ts.interval = '';
0018
0019     number = abs(ts.startT.utc_epoch_milli-ts.endT.utc_epoch_milli);
0020
0021     if (ts.endT.utc_epoch_milli-ts.startT.utc_epoch_milli < 0)
0022         ts.interval = '-';
0023     end
0024
0025     form = java.text.SimpleDateFormat;
0026     form.setTimeZone(java.util.TimeZone.getTimeZone('UTC'));
0027
0028     form.applyPattern('yyyy')
0029     num_y = char(form.format(java.util.Date(number)));
0030     num_y = str2double(num_y);
0031     num_y = num_y - 1970;
0032     num_y = sprintf('%02d', num_y);
0033
0034     form.applyLocalizedPattern('DD')
0035     num_d = char(form.format(java.util.Date(number)));
0036     num_d = sprintf('%02d', str2double(num_d)-1);
0037
0038     form.applyLocalizedPattern('HH')
0039     num_h = char(form.format(java.util.Date(number)));
0040
0041     form.applyLocalizedPattern('mm')
0042     num_m = char(form.format(java.util.Date(number)));
0043
0044     form.applyLocalizedPattern('ss')
0045     num_s = char(form.format(java.util.Date(number)));
0046
0047     form.applyLocalizedPattern('SSS')
0048     num_milli_s = char(form.format(java.util.Date(number)));
0049
0050     if ~strcmp(num_y, '00')
0051         ts.interval = [ts.interval num_y ' Years '];
0052     end
0053
0054     if ~strcmp(num_d, '00')
0055         ts.interval = [ts.interval num_d ' Days '];
0056     end
0057
0058     if ~strcmp(num_h, '00')
0059         ts.interval = [ts.interval num_h ' Hours '];
0060     end
0061
0062     if ~strcmp(num_m, '00')
0063         ts.interval = [ts.interval num_m ' Minutes '];
0064     end
0065
0066     if ~strcmp(num_s, '00')
0067         ts.interval = [ts.interval num_s ' Seconds '];
0068     end
0069
0070     if ~strcmp(num_milli_s, '000')
0071         ts.interval = [ts.interval num_milli_s ' Milliseconds '];
0072     end
0073 end
0074
0075

```


fromTimespanDef

PURPOSE ^

%%%

SYNOPSIS ^

```
function obj = fromTimespanDef(obj, pli)
```

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromTimespanDef
DESCRIPTION: Construct an timespan from start and end time
CALL:         ts = fromTimespanDef(ts, t1, t2)
INPUT:        ts = timespan-object
              t1 = start time (char or time-object)
              t2 = end   time (char or time-object)
HISTORY:      23-07-2008 Diepholz
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [computeInterval](#) COMPUTEINTERVAL compute the interval of the time span.

This function is called by:

 [timespan](#) TIMESPAN timespan object class constructor.

SOURCE CODE ^

```
0001 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0002 %
0003 % FUNCTION:      fromTimespanDef
0004 %
0005 % DESCRIPTION: Construct an timespan from start and end time
```



```

0006 %
0007 % CALL:      ts = fromTimespanDef(ts, t1, t2)
0008 %
0009 % INPUT:      ts = timespan-object
0010 %            t1 = start time (char or time-object)
0011 %            t2 = end   time (char or time-object)
0012 %
0013 % HISTORY:    23-07-2008 Diepholz
0014 %            Creation
0015 %
0016 %%%%%%%%%%%
0017 function obj = fromTimespanDef(obj, pli)
0018
0019     VERSION = '$Id: fromTimespanDef.m,v 1.3 2008/09/04 13:37:15 ingo Exp $';
0020     % get timespan info
0021     ii = timespan.getInfo('timespan', 'From Timespan Definition');
0022
0023     % Set the method version string in the minfo object
0024     ii.setMversion([VERSION '-->' ii.mversion]);
0025
0026     t1 = pli.find('startT');
0027     if isempty(t1)
0028         t1 = pli.find('start');
0029     end
0030     t2 = pli.find('endT');
0031     if isempty(t2)
0032         t2 = pli.find('end');
0033     end
0034
0035     % Add default values
0036     pl = combine(pli, ii.plist);
0037
0038     tf = pl.find('timeformat');
0039     tz = pl.find('timezone');
0040
0041
0042     if ischar(t1) || isnumeric(t1)
0043         t1 = time(t1);
0044     elseif ~isa(t1, 'time')
0045         error('### The start time must be a string or a time-object. But it is from the class [%s]', class(t1));
0046     end
0047
0048     if ischar(t2) || isnumeric(t2)
0049         t2 = time(t2);
0050     elseif ~isa(t2, 'time')
0051         error('### The end time must be a string or a time-object. But it is from the class [%s]', class(t2));
0052     end
0053
0054     obj.startT = time(t1);
0055     obj.endT   = time(t2);
0056
0057     % Adjust the 'timeformat' AND the 'timezone' of the end time to the start time.
0058     if isempty(tf)
0059         obj.setTimeformat(obj.startT.timeformat, 'internal');
0060     else
0061         obj.setTimeformat(tf, 'internal');
0062     end
0063     if isempty(tz)
0064         obj.setTimezone(obj.startT.timezone, 'internal');
0065     else
0066         obj.setTimezone(tz, 'internal');
0067     end
0068
0069     % Compute the interval
0070     obj = computeInterval(obj);
0071

```

Description of fromTimespanDef

```
0072 % Add history
0073 obj.addHistory(ii, pl, [], obj.hist);
0074
0075 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [timespan](#) TIMESPAN timespan object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034         % Update here the structure
0035     catch ME
0036         disp(varargin{1});
0037         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (time) above is not
from version 1.0')));
0041     end
0042     % Set the struct_ver to '1.1'
0043     struct_ver = '1.1';
0044 end
0045
0046 %%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%
0047
0048 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0049     strcmp(struct_ver, '1.1')
0050     % Set the struct_ver to '1.2'
0051     struct_ver = '1.1';
0052 end
0053
0054 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0055
0056 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0057     strcmp(struct_ver, '1.2')
0058     % Set the struct_ver to '1.3'
0059     struct_ver = '1.3';
0060 end
0061
0062 varargout{1} = obj_struct;
0063
0064
0065

```

epoch2str

PURPOSE ^

EPOCH2STR Returns computes the time sting of the epoch time stored in 'utc_epoch_milli' and 'created'.

SYNOPSIS ^

```
function time_str = epoch2str(tt)
```

DESCRIPTION ^

```
EPOCH2STR Returns computes the time sting of the epoch time stored in 'utc_epoch_milli' and
'created'.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: EPOCH2STR Returns computes the time sting of the epoch time
             stored in 'utc_epoch_milli' and 'created'.

CALL:
    string = epoch2str(tt);
    string = epoch2str(tt);
    string = epoch2str(tt)

VERSION:     $Id: epoch2str.m,v 1.2 2008/07/11 15:16:45 ingo Exp $

HISTORY:     04-04-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#) CHAR convert a time object into a string.

This function is called by:

 [display](#) DISPLAY overloads display functionality for time objects.

 [time](#) TIME time object class constructor.

SOURCE CODE ^

```
0001 function time_str = epoch2str(tt)
0002 % EPOCH2STR Returns computes the time sting of the epoch time stored in
0003 % 'utc_epoch_milli' and 'created'.
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: EPOCH2STR Returns computes the time sting of the epoch time
0007 % stored in 'utc_epoch_milli' and 'created'.
0008 %
0009 % CALL:
0010 %     string = epoch2str(tt);
0011 %     string = epoch2str(tt);
0012 %     string = epoch2str(tt)
0013 %
0014 % VERSION:     $Id: epoch2str.m,v 1.2 2008/07/11 15:16:45 ingo Exp $
0015 %
0016 % HISTORY:     04-04-2008 Diepholz
0017 %              Creation
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 %
```

```

0020 java_tf = tt.timeformat;
0021 for kk = 1:length(java_tf)
0022     switch java_tf(kk)
0023     case 'm'
0024         java_tf(kk) = 'M';
0025     case 'M'
0026         java_tf(kk) = 'm';
0027     case 'S'
0028         java_tf(kk) = 's';
0029     case 'F'
0030         java_tf(kk) = 'S';
0031     case 'P'
0032         if java_tf(kk+1) == 'M'
0033             java_tf(kk) = 'a';
0034             java_tf(kk+1) = 'a';
0035         end
0036     end
0037 end
0038
0039 t_format = java.text.SimpleDateFormat(java_tf);
0040 t_format.setTimeZone(tt.timezone);
0041 time_str = char(t_format.format(tt.utc_epoch_milli));
0042
0043 java_tf = getappdata(0, 'time_format_str');
0044 for kk = 1:length(java_tf)
0045     switch java_tf(kk)
0046     case 'm'
0047         java_tf(kk) = 'M';
0048     case 'M'
0049         java_tf(kk) = 'm';
0050     case 'S'
0051         java_tf(kk) = 's';
0052     case 'F'
0053         java_tf(kk) = 'S';
0054     end
0055 end
0056 t_format = java.text.SimpleDateFormat(java_tf);
0057 t_format.setTimeZone(tt.timezone);

```

fromTimeString

PURPOSE ^

FROMTIMESTRING creates a time object from a time string.

SYNOPSIS ^

function varargout = fromTimeString(varargin)

DESCRIPTION ^

```
FROMTIMESTRING creates a time object from a time string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromTimeString

DESCRIPTION: Construct a time object from a string value

CALL:         tt = fromTimeString(tt, plist)

INPUT:        tt:      input time object
               plist:  plist object which must contain the key 'time_str'

EXAMPLES:     Some examples for the value of the key 'time_str'
               '2008-06-29 14:00:00.000'
               '14:00:00.000 2008-06-29'
               '14:00:00 29-06-2008'
               '14:00:00'

HISTORY:      07-05-2007 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SOURCE CODE ^

```
0001 % FROMTIMESTRING creates a time object from a time string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromTimeString
0005 %
0006 % DESCRIPTION: Construct a time object from a string value
0007 %
0008 % CALL:         tt = fromTimeString(tt, plist)
0009 %
0010 % INPUT:        tt:      input time object
0011 %               plist:  plist object which must contain the key 'time_str'
0012 %
0013 % EXAMPLES:     Some examples for the value of the key 'time_str'
0014 %               '2008-06-29 14:00:00.000'
0015 %               '14:00:00.000 2008-06-29'
0016 %               '14:00:00 29-06-2008'
0017 %               '14:00:00'
0018 %
0019 % HISTORY:      07-05-2007 Hewitson
```

```

0020 %                               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = fromTimeString(varargin)
0025
0026     if nargin ~= 2
0027         error('### Unknown number of inputs.');
```

```

0028     end
0029
0030     % Get inputs
0031     tt = varargin{1};
0032     pli = varargin{2};
0033
0034     %% decide whether we modify the time-object, or create a new one.
0035     tt = copy(tt, nargin);
0036
0037     % Get the default plist
0038     def_pl = time.getInfo('time', 'From Time String').plists;
0039
0040     % combine input-plist and default-plist
0041     pl = combine(pli, def_pl);
0042
0043     % set values
0044     tt.setTimeformat(find(pl, 'timeformat'), 'internal');
0045     tt.setTimezone  {find(pl, 'timezone'), 'internal'};
0046     tt.setTime_str  {find(pl, 'time_str'), 'internal'};
0047
0048     % Sets again the user defined timeformat
0049     if find(pli, 'timeformat')
0050         tt.setTimeformat(find(pli, 'timeformat'), 'internal');
0051     end
0052
0053     varargout{1} = tt;

```


fromTimeValue

PURPOSE ^

FROMTIMEVALUE creates a time object from unix epoch time.

SYNOPSIS ^

function varargout = fromTimeValue(varargin)

DESCRIPTION ^

```
FROMTIMEVALUE creates a time object from unix epoch time.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromTimeString

DESCRIPTION: Construct a time object from unix epoch time.

CALL:         tt = fromTimeValue(tt, plist)

INPUT:        tt:      input time object
               plist:  plist object which must contain the key 'utc_epoch_milli'

EXAMPLES:     Some examples for the value of the key 'time_str'
               '2008-06-29 14:00:00.000'
               '14:00:00.000 2008-06-29'
               '14:00:00 29-06-2008'
               '14:00:00'

HISTORY:      07-05-2007 Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of time objects depending of the second input

This function is called by:

 [time](#) TIME time object class constructor.

SOURCE CODE ^

```
0001 % FROMTIMEVALUE creates a time object from unix epoch time.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromTimeString
0005 %
0006 % DESCRIPTION: Construct a time object from unix epoch time.
0007 %
0008 % CALL:         tt = fromTimeValue(tt, plist)
0009 %
0010 % INPUT:        tt:      input time object
0011 %               plist:  plist object which must contain the key 'utc_epoch_milli'
0012 %
0013 % EXAMPLES:     Some examples for the value of the key 'time_str'
0014 %               '2008-06-29 14:00:00.000'
0015 %               '14:00:00.000 2008-06-29'
0016 %               '14:00:00 29-06-2008'
0017 %               '14:00:00'
0018 %
0019 % HISTORY:      07-05-2007 Hewitson
```

```
0020 %                               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0023
0024 function varargout = fromTimeValue(varargin)
0025
0026     if nargin ~= 2
0027         error('### Unknown number of inputs.');
```

0028 end

0029

0030 % Get inputs

0031 tt = varargin{1};

0032 pli = varargin{2};

0033

0034 %% decide whether we modify the time-object, or create a new one.

0035 tt = copy(tt, nargout);

0036

0037 % Get the default plist

0038 def_pl = time.getInfo('time', 'From Time Definition').plist;

0039

0040 % combine input-plist and default-plist

0041 pl = combine(pli, def_pl);

0042

0043 % set values

0044 tt.setEpochtime (find(pl, 'utc_epoch_milli'), 'internal');

0045 tt.setTimeformat (find(pl, 'timeformat'), 'internal');

0046 tt.setTimezone (find(pl, 'timezone'), 'internal');

0047

0048 varargout{1} = tt;

parse_time_string

PURPOSE ^

PRASE_TIME_STRING parses the time string.

SYNOPSIS ^

function varargout = parse_time_string(varargin)

DESCRIPTION ^

```
PRASE_TIME_STRING parses the time string.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      parse_time_string

DESCRIPTION:   PRASE_TIME_STRING parses the time string and returns the
               corresponding MATLAB and JAVA format.

CALL:         tt = parse_time_string(time_str)

INPUT:        time_str: time string

EXAMPLES:     Some examples for the 'time_str'
               '2008-06-29 14:00:00.000'
               '14:00:00.000 2008-06-29'
               '14:00:00 29-06-2008'
               '14:00:00'

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [time](#) TIME time object class constructor.

SOURCE CODE ^

```
0001 % PRASE_TIME_STRING parses the time string.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      parse_time_string
0005 %
0006 % DESCRIPTION:   PRASE_TIME_STRING parses the time string and returns the
0007 %               corresponding MATLAB and JAVA format.
0008 %
0009 % CALL:         tt = parse_time_string(time_str)
0010 %
0011 % INPUT:        time_str: time string
0012 %
0013 % EXAMPLES:     Some examples for the 'time_str'
0014 %               '2008-06-29 14:00:00.000'
0015 %               '14:00:00.000 2008-06-29'
0016 %               '14:00:00 29-06-2008'
0017 %               '14:00:00'
0018 %
0019 % HISTORY:      07-07-2008 Diepholz
0020 %               Creation
0021 %
0022 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

0023
0024 function varargout = parse_time_string(varargin)
0025 % PARSE_TIME_STRING into
0026
0027 % 'match expression' 'JAVA date format' 'MATLAB date format'
0028 parse = { '\d{2}:\d{2}:\d{2}', 'HH:mm:ss', 'HH:MM:SS'; ...
0029           '\d{2}:\d{2}', 'mm:ss', 'MM:SS'; ...
0030           '\d{2}-\d{2}-\d{4}', 'dd-MM-yyyy', 'dd-mm-yyyy'; ...
0031           '\d{2}.\d{2}.\d{4}', 'dd.MM.yyyy', 'dd.mm.yyyy'; ...
0032           '\d{4}-\d{2}-\d{2}', 'yyyy-MM-dd', 'yyyy-mm-dd'; ...
0033           '\d{4}.\d{2}.\d{2}', 'yyyy.MM.dd', 'yyyy.mm.dd'; ...
0034           '\d{2}-\d{2}', 'MM-dd', 'mm-dd'; ...
0035           '\.\d{1,3}', '.SSS', '.FFF'};
0036
0037 ret_java = varargin{1};
0038 ret_matlab = varargin{1};
0039
0040 for ii = 1:size(parse,1)
0041
0042     curr_parse = parse{ii,1};
0043     [match] = regexp(ret_java, curr_parse, 'match');
0044
0045     if ~isempty(match) && length(match) == 1
0046         replace_java = parse{ii,2};
0047         replace_matlab = parse{ii,3};
0048         ret_java = strrep(ret_java, match{1}, replace_java);
0049         ret_matlab = strrep(ret_matlab, match{1}, replace_matlab);
0050     end
0051 end
0052
0053 % Set outputs
0054 varargout{1} = ret_java;
0055 varargout{2} = ret_matlab;
0056
0057 end
0058 % END
0059

```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [time](#) TIME time object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035         obj_struct = rmfield(obj_struct, 'name');
0036         obj_struct = rmfield(obj_struct, 'created');
0037         obj_struct = rmfield(obj_struct, 'plist');
0038
0039         if isstruct(obj_struct.timeformat)
0040             obj_struct.timeformat = obj_struct.timeformat.format_str;
0041         end
0042
0043     catch ME
0044         disp(varargin{1});
0045         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (time) above is not
0046 from version 1.0')));
0047     end
0048
0049     % Set the struct_ver to '1.1'
0050     struct_ver = '1.1';
0051 end
0052
0053 %%%%%%%%%% Update version '1.1 -> 1.2' %%%%%%%%%%
0054
0055 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0056     strcmp(struct_ver, '1.1')
0057
0058     % Set the struct_ver to '1.2'
0059     struct_ver = '1.1';
0060
0061 end
0062
0063 %%%%%%%%%% Update version '1.2 -> 1.3' %%%%%%%%%%
0064
0065 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0066     strcmp(struct_ver, '1.2')
0067
0068     % Set the struct_ver to '1.3'
0069     struct_ver = '1.3';
0070
0071 end
0072
0073 varargout{1} = obj_struct;
0074

```

fq2ri

PURPOSE ^

FQ2RI Convert frequency/Q pole/zero representation

SYNOPSIS ^

function ri= fq2ri(f0, Q)

DESCRIPTION ^

FQ2RI Convert frequency/Q pole/zero representation
DESCRIPTION: FQ2RI Convert frequency/Q pole/zero representation into real
and imaginary conjugate pairs.
Returns [ri conj(ri)]
CALL: ri= fq2ri(f0, Q)
VERSION: \$Id: fq2ri.m,v 1.3 2008/07/11 17:34:47 ingo Exp \$
HISTORY: 02-04-2007 M Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:
This function is called by:
🐘 pz PZ is the ltpda class that provides a common definition of poles and zeros.

SOURCE CODE ^

0001 % FQ2RI Convert frequency/Q pole/zero representation
0002 %
0003 %
0004 % DESCRIPTION: FQ2RI Convert frequency/Q pole/zero representation into real
0005 % and imaginary conjugate pairs.
0006 % Returns [ri conj(ri)]
0007 %
0008 % CALL: ri= fq2ri(f0, Q)
0009 %
0010 % VERSION: \$Id: fq2ri.m,v 1.3 2008/07/11 17:34:47 ingo Exp \$
0011 %
0012 % HISTORY: 02-04-2007 M Hewitson
0013 % Creation
0014 %
0015 %
0016 %
0017 function ri= fq2ri(f0, Q)
0018
0019 if(nargin==0)
0020 disp('usage: ri = fq2ri2(f0, Q)')
0021 return
0022 elseif(nargin==1 || nargin==2 && isnan(Q))
0023 ri = -(1/(2*pi*f0));
0024 elseif(nargin==2 && Q>=0)
0025 if Q < 0.5
0026 warning('!!! Q < 0.5! Splitting to two real poles;')
0027 ri = [-(1/(2*pi*f0)) -(1/(2*pi*f0))];
0028 else
0029 w0 = 2*pi*f0;

```
0030      G    = 1/(w0*w0);
0031      re    = G.*-w0/(2*Q);
0032      im    = G.*w0*sqrt(4*Q*Q-1)/(2*Q);
0033      tri   = complex(re,im);
0034      ri    = [tri; conj(tri)];
0035      end
0036  end
0037
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ri2fq

PURPOSE ^

RI2FQ Convert complx pole/zero into frequency/Q pole/zero representation.

SYNOPSIS ^

function [f0, q]= ri2fq(c)

DESCRIPTION ^

RI2FQ Convert complx pole/zero into frequency/Q pole/zero representation.
DESCRIPTION: RI2FQ Convert complx pole/zero into frequency/Q pole/zero representation.
CALL: [f0, q]= ri2fq(c)
VERSION: \$Id: ri2fq.m,v 1.2 2008/07/11 17:34:47 ingo Exp \$
HISTORY: 03-12-2003 M Hewitson Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 pz PZ is the ltpda class that provides a common definition of poles and zeros.

SOURCE CODE ^

```
0001 % RI2FQ Convert complx pole/zero into frequency/Q pole/zero representation.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RI2FQ Convert complx pole/zero into frequency/Q pole/zero
0005 % representation.
0006 %
0007 % CALL: [f0, q]= ri2fq(c)
0008 %
0009 % VERSION: $Id: ri2fq.m,v 1.2 2008/07/11 17:34:47 ingo Exp $
0010 %
0011 % HISTORY: 03-12-2003 M Hewitson
0012 % Creation
0013 %
0014 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0015
0016 function [f0, q]= ri2fq(c)
0017
0018 if(nargin==0)
0019     disp('usage: [f0, q]= ri2fq(c)');
0020     return
0021 end
0022
0023 a = real(c(1));
0024 b = imag(c(1));
0025
0026 % f0 = -a*sqrt(b^2/a^2 + 1) / 2 / pi;
0027 % q = sqrt(b^2/a^2 +1)/2;
0028
0029 k = -a*sqrt(b^2/a^2 + 1) * 2 * pi;
0030 f0 = 1/k;
```

```
0031    q = sqrt(b^2/a^2 +1)/2;  
0032  
0033    % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setF

PURPOSE ^

SETF Set the property 'f'

SYNOPSIS ^

function ii = setF(ii, val)

DESCRIPTION ^

```
SETF Set the property 'f'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETF Set the property 'f' and computes 'ri'

CALL:      obj = obj.setF(1);
           obj = setF(obj, 1);

INPUTS:    obj - is a pz object

VERSION:    $Id: setF.m,v 1.1 2008/07/11 17:34:47 ingo Exp $

HISTORY:    27-05-2008 Hewitson
           Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of pz objects depending of the second input

This function is called by:

 [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

SOURCE CODE ^

```
0001 % SETF Set the property 'f'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETF Set the property 'f' and computes 'ri'
0005 %
0006 % CALL:      obj = obj.setF(1);
0007 %           obj = setF(obj, 1);
0008 %
0009 % INPUTS:    obj - is a pz object
0010 %
0011 % VERSION:    $Id: setF.m,v 1.1 2008/07/11 17:34:47 ingo Exp $
0012 %
0013 % HISTORY:    27-05-2008 Hewitson
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function ii = setF(ii, val)
0019
0020 %%% decide whether we modify the pz-object, or create a new one.
0021 ii = copy(ii, nargin);
0022
0023 %%% set 'f'
0024 ii.f = val;
0025 %%% Then compute and set ri
```

```
0026   ii.ri = pz.fq2ri(ii.f, ii.q);  
0027 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setQ

PURPOSE ^

SETQ Set the property 'q'

SYNOPSIS ^

function ii = setQ(ii, val)

DESCRIPTION ^

SETQ Set the property 'q'

DESCRIPTION: SETQ Set the property 'q' and computes 'ri'

CALL: obj = obj.setQ(1);
obj = setQ(obj, 1);

INPUTS: obj - is a pz object

VERSION: \$Id: setQ.m,v 1.1 2008/07/11 17:34:47 ingo Exp \$

HISTORY: 27-05-2008 Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of pz objects depending of the second input

This function is called by:

 [pz](#) PZ is the ltpda class that provides a common definition of poles and zeros.

SOURCE CODE ^

```
0001 % SETQ Set the property 'q'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETQ Set the property 'q' and computes 'ri'
0005 %
0006 % CALL:      obj = obj.setQ(1);
0007 %           obj = setQ(obj, 1);
0008 %
0009 % INPUTS:    obj - is a pz object
0010 %
0011 % VERSION:   $Id: setQ.m,v 1.1 2008/07/11 17:34:47 ingo Exp $
0012 %
0013 % HISTORY:   27-05-2008 Hewitson
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function ii = setQ(ii, val)
0019
0020 %%% decide whether we modify the pz-object, or create a new one.
0021 ii = copy(ii, nargout);
0022
0023 %%% set 'q'
0024 ii.q = val;
0025 %%% Then compute and set ri
```

```
0026    ii.ri = pz.fq2ri(ii.f, ii.q);
0027
0028    %%% did we split to two real poles?
0029    if ii.q < 0.5
0030        ri = ii.ri;
0031        ii.ri = ri(1);
0032        ii(2).f = ii.f;
0033        ii(2).q = NaN;
0034        ii(2).ri = ri(1);
0035    end
0036 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

setRI

PURPOSE ^

SETRI Set the property 'ri' and computes 'f' and 'q'

SYNOPSIS ^

function ii = setRI(ii, val)

DESCRIPTION ^

SETRI Set the property 'ri' and computes 'f' and 'q'

DESCRIPTION: SETRI Set the property 'ri' and computes 'f' and 'q'

CALL: obj = obj.setRI(val);
obj = setRI(obj, val);

INPUTS: obj - is a pz object

VERSION: \$Id: setRI.m,v 1.1 2008/07/11 17:34:47 ingo Exp \$

HISTORY: 27-05-2008 Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 **copy** COPY Make copy of pz objects depending of the second input

This function is called by:

 **pz** PZ is the ltpda class that provides a common definition of poles and zeros.

SOURCE CODE ^

```
0001 % SETRI Set the property 'ri' and computes 'f' and 'q'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETRI Set the property 'ri' and computes 'f' and 'q'
0005 %
0006 % CALL:      obj = obj.setRI(val);
0007 %           obj = setRI(obj, val);
0008 %
0009 % INPUTS:    obj - is a pz object
0010 %
0011 % VERSION:   $Id: setRI.m,v 1.1 2008/07/11 17:34:47 ingo Exp $
0012 %
0013 % HISTORY:   27-05-2008 Hewitson
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function ii = setRI(ii, val)
0019
0020 %%% decide whether we modify the pz-object, or create a new one.
0021 ii = copy(ii, nargin);
0022
0023 if numel(val) == 1
0024     % add conjugate
0025     val = [val conj(val)];
```

```
0026 elseif numel(val) == 2
0027     if val(1) ~= conj(val(2))
0028         error('### Please enter a conjugate pair to specify a complex pole.');
```

```
0029     end
0030 else
0031     error('### A pole/zero must be defined with a conjugate pair or a single complex
number');
0032 end
0033
0034 %%% set 'ri'
0035 ii.ri = val;
0036 %%% Then compute and set f and Q
0037 [f,q] = pz.ri2fq(ii.ri);
0038 ii.f = f;
0039 ii.q = q;
0040 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.2 2008/07/11 17:34:47 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 pz PZ is the ltpda class that provides a common definition of poles and zeros.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.2 2008/07/11 17:34:47 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020     obj_struct = varargin{1};
0021     struct_ver = varargin{2};
0022
0023     % get the version of the current toolbox
0024     tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026     % get only the version string without the MATLAB version
0027     struct_ver = strtok(struct_ver);
0028
```

```
0029 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034
0035         obj_struct = rmfield(obj_struct, 'name');
0036         obj_struct = rmfield(obj_struct, 'created');
0037         obj_struct = rmfield(obj_struct, 'plist');
0038
0039     catch ME
0040         disp(varargin{1});
0041         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (pz) above is not
from version 1.0')));
0042     end
0043
0044     % Set the struct_ver to '1.1'
0045     struct_ver = '1.1';
0046
0047 end
0048
0049 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.1 ->1.2' %%%%%%%%%%%%%%%%%%%%%%%%%%
0050
0051 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0052     strcmp(struct_ver, '1.1')
0053
0054     % Set the struct_ver to '1.2'
0055     struct_ver = '1.1';
0056
0057 end
0058
0059 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%%%%%%%%%%%%%%%%%
0060
0061 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0062     strcmp(struct_ver, '1.2')
0063
0064     % Set the struct_ver to '1.3'
0065     struct_ver = '1.3';
0066
0067 end
0068
0069 varargout{1} = obj_struct;
0070
0071
```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.2 2008/09/05 14:16:11 hewitson Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [minfo](#) MINFO a helper class for LTPDA methods.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.2 2008/09/05 14:16:11 hewitson Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```
0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034     catch ME
0035         disp(varargin{1});
0036         throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (minfo) above is not
from version 1.0')));
0039     end
0040
0041     % Set the struct_ver to '1.9.1'
0042     struct_ver = '1.9.1';
0043
0044 end
0045
0046 %%%%%%%%%% Update version '1.9.1' ->'1.9.2' %%%%%%%%%%
0047
0048 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0049     strcmp(struct_ver, '1.9.1')
0050
0051     % Set the struct_ver to '1.2'
0052     struct_ver = '1.9.2';
0053
0054 end
0055
0056 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0057
0058 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0059     strcmp(struct_ver, '1.2')
0060
0061     % Set the struct_ver to '1.3'
0062     struct_ver = '1.3';
0063
0064 end
0065
0066 varargout{1} = obj_struct;
0067
```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [history](#) HISTORY History object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034 try
0035     %%% methodInfo
0036     ii = minfo(obj_struct.name, 'ao', '', '', obj_struct.version, {''}, plist, 0, 0);
0037     obj_struct.methodInfo = ii;
0038
0039     %%% version
0040     obj_struct.version = obj_struct.consver;
0041
0042     %%% proctime
0043     obj_struct.proctime = obj_struct.created;
0044
0045     %%% methodInvars
0046     obj_struct.methodInvars = obj_struct.invars;
0047
0048     %%% plistUsed
0049     obj_struct.plistUsed = obj_struct.plist;
0050 catch ME
0051     disp(varargin{1});
0052     throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (history) above is not
from version 1.0')));
0053 end
0054
0055 % Set the struct_ver to '1.1'
0056 struct_ver = '1.1';
0057
0058 end
0059
0060 %%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%
0061
0062 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0063     strcmp(struct_ver, '1.1')
0064
0065     % Set the struct_ver to '1.2'
0066     struct_ver = '1.1';
0067
0068 end
0069
0070 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0071
0072 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0073     strcmp(struct_ver, '1.2')
0074
0075     % Set the struct_ver to '1.3'
0076     struct_ver = '1.3';
0077
0078 end
0079
0080 varargout{1} = obj_struct;
0081

```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.4 2008/08/22 14:17:26 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [provenance](#) PROVENANCE constructors for provenance class.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.4 2008/08/22 14:17:26 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020     obj_struct = varargin{1};
0021     struct_ver = varargin{2};
0022
0023     % get the version of the current toolbox
0024     tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026     % get only the version string without the MATLAB version
0027     struct_ver = strtok(struct_ver);
0028
```

```

0029  %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031  if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver)
0032
0033      try
0034          obj_struct.version = 'updater 1.0';
0035          obj_struct.symbolic_math_version = 'updateer 1.0';
0036      catch ME
0037          disp(varargin{1});
0038          throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (provenance) above is
not from version 1.0')));
0039      end
0040
0041      % Set the struct_ver to '1.1'
0042      struct_ver = '1.1';
0043
0044  end
0045
0046  %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%%%%%%%%%%%%%%%%%
0047
0048  if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0049      strcmp(struct_ver, '1.1')
0050
0051      % Set the struct_ver to '1.2'
0052      struct_ver = '1.1';
0053
0054  end
0055
0056  %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%%%%%%%%%%%%%%%%%
0057
0058  if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0059      strcmp(struct_ver, '1.2')
0060
0061      % Set the struct_ver to '1.3'
0062      struct_ver = '1.3';
0063
0064  end
0065
0066  varargout{1} = obj_struct;
0067

```


update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         [obj_struct, obj_class] = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.2 2008/09/05 14:16:11 hewitson Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [cdata](#) CDATA is the constant data class.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         [obj_struct, obj_class] = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.2 2008/09/05 14:16:11 hewitson Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.0 -> 1.9.1' %%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033     try
0034         obj_struct = rmfield(obj_struct, 'name');
0035     catch ME
0036         disp(varargin{1});
0037         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (cdata) above is not
from version 1.0')));
0041     end
0042
0043     % Set the struct_ver to '1.9.1'
0044     struct_ver = '1.9.1';
0045
0046 end
0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.9.1' ->'1.9.2' %%%%%%%%%%%%%%%%%%%%%%%%%%
0049
0050 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0051     strcmp(struct_ver, '1.9.1')
0052     % Check Y units
0053     try
0054         uo = unit(obj_struct.yunits);
0055     catch
0056         warning('!!! This XML file contains a fsdata object with unsupported y-units [%s].
Setting to empty.', obj_struct.yunits);
0058         obj_struct.yunits = '';
0059     end
0060
0061     % Check X units
0062     try
0063         uo = unit(obj_struct.xunits);
0064     catch
0065         warning('!!! This XML file contains a fsdata object with unsupported x-units [%s].
Setting to empty.', obj_struct.xunits);
0066         obj_struct.xunits = '';
0067     end
0068
0069     % Set the struct_ver to '1.9.2'
0070     struct_ver = '1.9.2';
0071
0072 end
0073
0074 %%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.9.2' ->'1.9.3' %%%%%%%%%%%%%%%%%%%%%%%%%%
0075
0076 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0077     strcmp(struct_ver, '1.9.2')
0078     % Set the struct_ver to '1.9.3'
0079     struct_ver = '1.9.3';
0080
0081
0082 end
0083
0084 varargout{1} = obj_struct;
0085

```

[Home](#) > [classes](#) > [@fsdata](#) > getFfromYFs.m

getFfromYFs

PURPOSE [^]

GETDSFROMYFS grows an evenly spaced frequency vector of N points for

SYNOPSIS [^]

function f = getFfromYFs(N,fs)

DESCRIPTION [^]

GETDSFROMYFS grows an evenly spaced frequency vector of N points for samplerate fs.

VERSION: \$Id: getFfromYFs.m,v 1.1 2008/06/11 13:11:41 hewitson Exp \$
M Hewitson 2008-06-11

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [fsdata](#) FSDATA frequency-series object class constructor.

SOURCE CODE [^]

```
0001 function f = getFfromYFs(N,fs)
0002 % GETDSFROMYFS grows an evenly spaced frequency vector of N points for
0003 % samplerate fs.
0004 %
0005 %
0006 % VERSION: $Id: getFfromYFs.m,v 1.1 2008/06/11 13:11:41 hewitson Exp $
0007 %
0008 % M Hewitson 2008-06-11
0009 %
0010 %
0011
0012 f = linspace(0, fs/2, N);
0013
0014
0015 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.3 2008/09/05 14:16:11 hewitson Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [fsdata](#) FSDATA frequency-series object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.3 2008/09/05 14:16:11 hewitson Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020     obj_struct = varargin{1};
0021     struct_ver = varargin{2};
0022
0023     % get the version of the current toolbox
0024     tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026     % get only the version string without the MATLAB version
0027     struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.0 -> 1.9.1' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034 try
0035     obj_struct = rmfield(obj_struct, 'name');
0036     obj_struct = rmfield(obj_struct, 'created');
0037     obj_struct.t0 = time(0);
0038
0039     if isempty(obj_struct.navs)
0040         obj_struct.navs = NaN;
0041     end
0042
0043     if isempty(obj_struct.fs)
0044         obj_struct.fs = NaN;
0045     end
0046
0047     if isempty(obj_struct.enbw)
0048         obj_struct.enbw = NaN;
0049     end
0050
0051
0052 catch ME
0053     disp(varargin{1});
0054     throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (fsdata) above is not
from version 1.0')));
0055 end
0056
0057 % Set the struct_ver to '1.9.1'
0058 struct_ver = '1.9.1';
0059
0060 end
0061
0062 %%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.9.1' ->'1.9.2' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063
0064 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0065     strcmp(struct_ver, '1.9.1')
0066
0067     % Make sure we have strings for the units
0068     obj_struct.xunits = char(obj_struct.xunits);
0069     obj_struct.yunits = char(obj_struct.yunits);
0070
0071     % We need to fix any strange old units
0072
0073     %- /\surdHz -> Hz^-0.5
0074     obj_struct.xunits = strrep(obj_struct.xunits, '\surdHz', ' Hz^-0.5');
0075     obj_struct.yunits = strrep(obj_struct.yunits, '\surdHz', ' Hz^-0.5');
0076
0077     % Check Y units
0078     try
0079         uo = unit(obj_struct.yunits);
0080     catch
0081         warning('!!! This XML file contains a fsdata object with unsupported y-units
[%s]. Setting to empty.', obj_struct.yunits);
0082         obj_struct.yunits = '';
0083     end
0084
0085     % Check X units
0086     try
0087         uo = unit(obj_struct.xunits);
0088     catch
0089         warning('!!! This XML file contains a fsdata object with unsupported x-units
[%s]. Setting to empty.', obj_struct.xunits);
0090         obj_struct.xunits = '';
0091     end
0092     % Set the struct_ver to '1.9.2'
0093     struct_ver = '1.9.1';
0094
0095 end
0096
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%% Update version '1.9.2' ->'1.9.3' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098
0099 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0100     strcmp(struct_ver, '1.9.2')
0101
0102     % Set the struct_ver to '1.9.3'
0103     struct_ver = '1.9.3';
0104
0105 end
0106
0107 varargout{1} = obj_struct;
0108 end
0109

```


data2D

PURPOSE ^

DATA2D is the abstract base class for 2-dimensional data objects.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
DATA2D is the abstract base class for 2-dimensional data objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DATA2D is the base class for 2-dimensional data objects. This is
              an abstract class.

SUPER CLASSES: ltpda_data < ltpda_nuo < ltpda_obj

SUB CLASSES:   xydata, tsdata, fsdata, data3D

PROPERTIES:

    Inherit Properties (read only)
        version - cvs-version string.

    Protected Properties (read only)
        xunits - units of the x-axis
        yunits - units of the y-axis
        x      - data values of the x-axis
        y      - data values of the y-axis

DATA2D METHODS:

    Public Methods
        char          - returns one character string which represents the object
        setXunits     - set the property 'xunits'
        setYunits     - set the property 'yunits'
        setX          - set the property 'x'
        setY          - set the property 'y'
        setXY         - set the property 'x' and 'y'
        getX          - get the property 'x'
        getY          - get the property 'y'
        applymethod   - applys the given method to the input 2D data
        applyoperator - applys the given operator to the two input data objects

VERSION:   $Id: data2D.m,v 1.23 2008/09/07 18:11:58 hewitson Exp $

HISTORY:   09-06-2008 Hewitson
              Creation.

SEE ALSO:  ltpda_data, fsdata, tsdata, xydata, data3D
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [applymethod](#) APPLYMETHOD applys the given method to the input 2D data.
-  [applyoperator](#) APPLYOPERATOR applys the given operator to the two input data objects.
-  [char](#) CHAR convert a ltpda_data-object into a string.
-  [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.
-  [getX](#) GETX Get the property 'x'.
-  [getY](#) GETY Get the property 'y'.

-  [setX](#) SETX Set the property 'x'.
-  [setXY](#) SETXY Set the property 'xy'.
-  [setXunits](#) SETXUNITS Set the property 'xunits'.
-  [setY](#) SETY Set the property 'y'.
-  [setYunits](#) SETYUNITS Set the property 'yunits'.

This function is called by:

-  [data2D](#) DATA2D is the abstract base class for 2-dimensional data objects.

SUBFUNCTIONS ^

-  [function obj = set.xunits\(obj, val\)](#)
-  [function obj = set.yunits\(obj, val\)](#)
-  [function obj = set.x\(obj, val\)](#)
-  [function obj = set.y\(obj, val\)](#)
-  [function obj = data2D\(varargin\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % DATA2D is the abstract base class for 2-dimensional data objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DATA2D is the base class for 2-dimensional data objects. This is
0005 %               an abstract class.
0006 %
0007 % SUPER CLASSES: ltpda_data < ltpda_nuo < ltpda_obj
0008 %
0009 % SUB CLASSES:   xydata, tsdata, fsdata, data3D
0010 %
0011 % PROPERTIES:
0012 %
0013 %     Inherit Properties (read only)
0014 %         version - cvs-version string.
0015 %
0016 %     Protected Properties (read only)
0017 %         xunits - units of the x-axis
0018 %         yunits - units of the y-axis
0019 %         x      - data values of the x-axis
0020 %         y      - data values of the y-axis
0021 %
0022 % DATA2D METHODS:
0023 %
0024 %     Public Methods
0025 %         char - returns one character string which represents the object
0026 %         setXunits - set the property 'xunits'
0027 %         setYunits - set the property 'yunits'
0028 %         setX      - set the property 'x'
0029 %         setY      - set the property 'y'
0030 %         setXY     - set the property 'x' and 'y'
0031 %         getX      - get the property 'x'
0032 %         getY      - get the property 'y'
0033 %         applymethod - applies the given method to the input 2D data
0034 %         applyoperator - applies the given operator to the two input data objects
0035 %
0036 % VERSION:  $Id: data2D.m,v 1.23 2008/09/07 18:11:58 hewitson Exp $
0037 %
0038 % HISTORY:   09-06-2008 Hewitson
0039 %             Creation.
0040 %
0041 % SEE ALSO:  ltpda_data, fsdata, tsdata, xydata, data3D
0042 %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044 %
0045 classdef data2D < ltpda_data
0046
```

```

0047
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049 %                                     Property definition                                     %
0050 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0051
0052 %----- Public (read/write) Properties -----
0053 properties
0054 end
0055
0056 %----- Protected read-only Properties -----
0057 properties
0058     xunits = unit;
0059     yunits = unit;
0060     x      = [];
0061     y      = [];
0062 end
0063
0064 %----- Private Properties -----
0065 properties (GetAccess = protected, SetAccess = protected)
0066 end
0067
0068 %----- Abstract Properties -----
0069 properties (Abstract = true, SetAccess = protected)
0070 end
0071
0072 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0073 %                                     Check property setting                                     %
0074 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075
0076 methods
0077     %--- Xunits
0078     function obj = set.xunits(obj, val)
0079         if ~ischar(val) && ~isa(val, 'unit')
0080             error('### The value for the property ''xunits'' must be a unit-object');
0081         end
0082         if ischar(val)
0083             obj.xunits = unit(val);
0084         elseif isa(val, 'unit')
0085             obj.xunits = val;
0086         else
0087             error('### The xunits value must be a unit or a string');
0088         end
0089     end
0090     %--- Yunits
0091     function obj = set.yunits(obj, val)
0092         if ~ischar(val) && ~isa(val, 'unit')
0093             error('### The value for the property ''yunits'' must be a unit-object');
0094         end
0095         if ischar(val)
0096             obj.yunits = unit(val);
0097         elseif isa(val, 'unit')
0098             obj.yunits = val;
0099         else
0100             error('### The yunits value must be a unit or a string');
0101         end
0102     end
0103     %--- X
0104     function obj = set.x(obj, val)
0105         if ~isempty(val)
0106             if ~isnumeric(val) || ~isvector(val)
0107                 error('### The value for the property ''x'' must be a numeric vector');
0108             end
0109         end
0110         obj.x = val;
0111
0112         sx = size(obj.x);
0113         sy = size(obj.y);
0114         if sx(1) == 1 && sy(1) ~= 1
0115             obj.x = obj.x.';
0116         end
0117         if sx(2) == 1 && sy(2) ~= 1
0118             obj.x = obj.x.';
0119         end
0120     end
0121     %--- Y
0122     function obj = set.y(obj, val)
0123         if ~isempty(val)
0124             if ~isnumeric(val) || ~isvector(val)
0125                 error('### The value for the property ''y'' must be a numeric vector');
0126             end
0127         end
0128     end
0129     if ~isempty(obj.y) && ((size(val,1) == 1 && size(obj.y,1) ~= 1) || (size(val,2) ==
1 && size(obj.y,2) ~= 1))
0130         val = val.';
0131     end

```

```
0132     obj.y = val;
0133
0134     sx = size(obj.x);
0135     sy = size(obj.y);
0136     if sx(1) == 1 && sy(1) ~= 1
0137         obj.x = obj.x.';
0138     end
0139     if sx(2) == 1 && sy(2) ~= 1
0140         obj.x = obj.x.';
0141     end
0142
0143 end
0144 end
0145
0146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0147 %                                     Constructor                                     %
0148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0149
0150 methods
0151     function obj = data2D(varargin)
0152
0153         %% Call superclass
0154         obj@ltpda_data(varargin{:});
0155
0156         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0157         % Set default values %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0158         if nargin == 1
0159             if isstruct(varargin{1})
0160                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0161                 % obj = data2D(struct) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0162
0163                 %% Set properties which are declared in this class
0164                 data2D_struct = varargin{1};
0165
0166                 obj.xunits = unit(data2D_struct.xunits);
0167                 obj.yunits = unit(data2D_struct.yunits);
0168                 obj.x      = data2D_struct.x;
0169                 obj.y      = data2D_struct.y;
0170             end
0171         end
0172
0173         %% Check the right size of x and y
0174         if ~isempty(obj.x)
0175             if ~isequal(size(obj.x), size(obj.y))
0176                 error('### The x and y containers must be the same size (or x must be
empty).');
0177             end
0178         end
0179     end % End constructor
0180
0181 end
0182
0183 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0184 %                                     Methods (public)                                     %
0185 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0186
0187 methods
0188     varargout = char(varargin)
0189     varargout = setXunits(varargin)
0190     varargout = setYunits(varargin)
0191     varargout = setX(varargin)
0192     varargout = setY(varargin)
0193     varargout = setXY(varargin)
0194     varargout = getX(varargin)
0195     varargout = getY(varargin)
0196
0197     varargout = applymethod(varargin)
0198     varargout = applyoperator(varargin)
0199 end
0200
0201 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0202 %                                     Methods (protected)                                     %
0203 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0204
0205 methods (Access = protected)
0206 end
0207
0208 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0209 %                                     Methods (public)                                     %
0210 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0211
0212 methods
0213 end
0214
0215 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0216 %                                     Methods (static)                                     %
0217 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
0217 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0218
0219 methods (Static)
0220
0221     function ii = getInfo(varargin)
0222         ii = utils.helper.generic_getInfo(varargin{:}, 'data2D');
0223     end
0224
0225     function out = VEROUT()
0226         out = '$Id: data2D.m,v 1.23 2008/09/07 18:11:58 hewitson Exp $';
0227     end
0228
0229     function out = SETS()
0230         out = {};
0231     end
0232
0233     function out = getDefaultPlist()
0234         out = [];
0235     end
0236
0237 end
0238
0239 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0240 %                                Methods (abstract)                                %
0241 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0242
0243 methods (Static)
0244 end
0245
0246 end
0247
```

[Home](#) > [classes](#) > [@tsdata](#) > fitfs.m

fitfs

PURPOSE [^]

FITFS computes the sample rate of the input data. If the min and max sample

SYNOPSIS [^]

```
function varargout = fitfs(varargin)
```

DESCRIPTION [^]

FITFS computes the sample rate of the input data. If the min and max sample intervals are within a certain tolerance (1%) of the median sample interval, the sample rate is returned as the median sample interval. If not, the sample rate is computed by fitting a grid to the input vertices. The fit is a two parameter fit for the start time and the sample rate. The MATLAB function lsqcurvefit is used to minimise

```
sum[ (ts - linspace(t0, (N-1)/fs, N))^2 ]
```

Call the function with

```
fs = fitfs(x)
```

or

```
[fs, t0] = fitfs(x)
```

or

```
[fs, t0, fitted] = fitfs(x)
```

where 'fitted' is a binary flag that tells whether the sample rate was fitted to the data. In other words, if fitted==true, then the data are considered to be unevenly sampled.

The fit for fs is constrained to be within 1 sigma of the median of the sample intervals. The fit for t0 is constrained to be within +/-1/fs.

In addition, the recovered sample rate is reduced to 3 significant figures.

M Hewitson 26-05-08

\$Id: fitfs.m,v 1.7 2008/09/04 14:14:02 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [tsdata](#) TSDATA time-series object class constructor.

SUBFUNCTIONS [^]

 [function t = tgrid\(x, xdata\)](#)

SOURCE CODE [^]

```
0001 % FITFS computes the sample rate of the input data. If the min and max sample
0002 % intervals are within a certain tolerance (1%) of the median sample interval,
0003 % the sample rate is returned as the median sample interval. If not, the sample
0004 % rate is computed by fitting a grid to the input vertices. The fit is a two
0005 % parameter fit for the start time and the sample rate. The MATLAB function
0006 % lsqcurvefit is used to minimise
0007 %
0008 %     sum[ (ts - linspace(t0, (N-1)/fs, N))^2 ]
```

```

0009 %
0010 % Call the function with
0011 %
0012 %     fs = fitfs(x)
0013 % or
0014 %     [fs, t0] = fitfs(x)
0015 % or
0016 %     [fs, t0, fitted] = fitfs(x)
0017 % where 'fitted' is a binary flag that tells whether the sample rate was
0018 % fitted to the data. In other words, if fitted==true, then the data are
0019 % considered to be unevenly sampled.
0020 %
0021 % The fit for fs is constrained to be within 1 sigma of the median of the
0022 % sample intervals. The fit for t0 is constrained to be within +/-1/fs.
0023 %
0024 % In addition, the recovered sample rate is reduced to 3 significant
0025 % figures.
0026 %
0027 % M Hewitson 26-05-08
0028 %
0029 % $Id: fitfs.m,v 1.7 2008/09/04 14:14:02 hewitson Exp $
0030 %
0031
0032 function varargout = fitfs(varargin)
0033
0034 import utils.const.*
0035
0036 tol = 1e-2;
0037
0038 % Get input vertices
0039 xi = varargin{1};
0040 % reshape x to match with linspace output
0041 ss = size(xi);
0042 if ss(1) > ss(2)
0043     xi = xi.';
0044 end
0045 d = diff(xi);
0046
0047 % Some statistics on x
0048 medd = median(d);
0049 mind = min(d);
0050 maxd = max(d);
0051 % Initial guess at sample rate
0052 ifs = 1.0/medd;
0053
0054 % Do we need a fit?
0055 fitted = false;
0056 if abs(maxd-medd)/medd>tol || abs(mind-medd)/medd>tol
0057
0058     utils.helper.msg(msg.OPROC1, 'fitting t0 and sample rate for unevenly sampled
0059 data.');
```

```

0060 %     % We need sigma for fit constraints
0061 %     sigma = std(1./d);
0062 %
0063 %     % Initial guess
0064 %     x0 = [xi(1) ifs];
0065 %
0066 %     lb = [-1/ifs (1-sigma)*ifs];
0067 %     ub = [1/ifs (1+sigma)*ifs];
0068 %
0069 %     % 'Display','iter',...
0070 %     options = optimset('MaxFunEvals', 1e3, ...
0071 %         'TolFun', 1e-20, 'TolX', 1e-20, 'MaxIter', 1e3);
0072 %
0073 %     x = lsqcurvefit(@tgrid, x0, xi, xi, lb, ub, options);
0074 %
0075 %     utils.helper.msg(msg.OPROC2, 'fit returned: %g s, %g Hz', x(1), x(2));
0076 % Return only 3 significant figures
0077 %     x(1) = eval(sprintf('%0.8g', x(1)));
0078 %     x(2) = eval(sprintf('%0.8g', x(2)));
0079 %     fitted = true;
0080 % We return the median and the intial sample time
0081 %     x(1) = 0; %eval(sprintf('%0.8g', xi(1)));
0082 %     x(2) = eval(sprintf('%0.8g', ifs));
0083 %     utils.helper.msg(msg.OPROC2, 'setting: %g s, %g Hz', x(1), x(2));
0084 else
0085 % We return the median and the intial sample time
0086 %     x(1) = eval(sprintf('%0.8g', xi(1)));
0087 %     x(2) = eval(sprintf('%0.8g', ifs));
0088 end
0089
0090
0091 % Set outputs
0092 if nargout == 0
0093     disp(x);

```

```
0094 elseif nargout == 1
0095     varargout{1} = x(2);
0096 elseif nargout == 2
0097     varargout{1} = x(2);
0098     varargout{2} = x(1);
0099 elseif nargout == 3
0100     varargout{1} = x(2);
0101     varargout{2} = x(1);
0102     varargout{3} = fitted;
0103 else
0104     error('### Unknown number of outputs.');
```

```
0105 end
0106
0107 %-----
0108 % Target function to minimise
0109 %-----
0110 function t = tgrid(x, xdata)
0111     t0 = x(1);
0112     fs = x(2);
0113     N = length(xdata);
0114     t = linspace(t0, (N-1)/fs, N);
0115 end
0116
0117 end % End fitfs
0118 % END
```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.3 2008/09/05 14:16:11 hewitson Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [tsdata](#) TSDATA time-series object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.3 2008/09/05 14:16:11 hewitson Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020     obj_struct = varargin{1};
0021     struct_ver = varargin{2};
0022
0023     % get the version of the current toolbox
0024     tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026     % get only the version string without the MATLAB version
0027     struct_ver = strtok(struct_ver);
0028
```



```

0029 %%%%%%%%%%%%%%% Update version '1.0 -> 1.9.1' %%%%%%%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035         obj_struct = rmfield(obj_struct, 'name');
0036         obj_struct = rmfield(obj_struct, 'created');
0037
0038
0039         if isempty(obj_struct.fs)
0040             obj_struct.fs = NaN;
0041         end
0042
0043     catch ME
0044         disp(varargin{1});
0045         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (fsdata) above is not
from version 1.0')));
0046     end
0047
0048     % Set the struct_ver to '1.9.1'
0049     struct_ver = '1.9.1';
0050
0051 end
0052
0053 %%%%%%%%%%%%%%% Update version '1.9.1' ->'1.9.2' %%%%%%%%%%%%%%%
0054
0055 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0056     strcmp(struct_ver, '1.9.1')
0057
0058
0059     % Check Y units
0060     try
0061         uo = unit(obj_struct.yunits);
0062     catch
0063         warning('!!! This XML file contains a fsdata object with unsupported y-units
[%s]. Setting to empty.', obj_struct.yunits);
0064         obj_struct.yunits = '';
0065     end
0066
0067     % Check X units
0068     try
0069         uo = unit(obj_struct.xunits);
0070     catch
0071         warning('!!! This XML file contains a fsdata object with unsupported x-units
[%s]. Setting to empty.', obj_struct.xunits);
0072         obj_struct.xunits = '';
0073     end
0074
0075     % Set the struct_ver to '1.9.2'
0076     struct_ver = '1.9.2';
0077
0078 end
0079
0080 %%%%%%%%%%%%%%% Update version '1.9.2' ->'1.9.3' %%%%%%%%%%%%%%%
0081
0082 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0083     strcmp(struct_ver, '1.9.2')
0084
0085     % Set the struct_ver to '1.9.3'
0086     struct_ver = '1.9.3';
0087
0088 end
0089
0090 varargout{1} = obj_struct;
0091 end
0092
0093

```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.2 2008/09/05 14:16:11 hewitson Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [xydata](#) XYDATA X-Y data object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.2 2008/09/05 14:16:11 hewitson Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.9.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035         obj_struct = rmfield(obj_struct, 'name');
0036         obj_struct = rmfield(obj_struct, 'created');
0037     catch ME
0038         disp(varargin{1});
0039         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (fsdata) above is not
from version 1.0')));
0040     end
0041
0042     % Set the struct_ver to '1.9.1'
0043     struct_ver = '1.9.1';
0044
0045 end
0046
0047 %%%%%%%%%% Update version '1.9.1' -> '1.9.2' %%%%%%%%%%
0048
0049 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0050     strcmp(struct_ver, '1.9.1')
0051
0052     % Check Y units
0053     try
0054         uo = unit(obj_struct.yunits);
0055     catch
0056         warning('!!! This XML file contains a fsdata object with unsupported y-units
[%s]. Setting to empty.', obj_struct.yunits);
0057         obj_struct.yunits = '';
0058     end
0059
0060     % Check X units
0061     try
0062         uo = unit(obj_struct.xunits);
0063     catch
0064         warning('!!! This XML file contains a fsdata object with unsupported x-units
[%s]. Setting to empty.', obj_struct.xunits);
0065         obj_struct.xunits = '';
0066     end
0067
0068     % Set the struct_ver to '1.9.2'
0069     struct_ver = '1.9.2';
0070
0071 end
0072
0073 %%%%%%%%%% Update version '1.9.2' -> '1.9.3' %%%%%%%%%%
0074
0075 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0076     strcmp(struct_ver, '1.9.2')
0077
0078     % Set the struct_ver to '1.9.3'
0079     struct_ver = '1.9.3';
0080
0081 end
0082
0083 varargout{1} = obj_struct;
0084

```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

```
UPDATE_STRUCT update the input structure to the current ltpda version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      update_struct

DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
               ltpda version

CALL:         obj_struct = update_struct(obj_struct, version_str);

VERSION:      $Id: update_struct.m,v 1.1 2008/07/25 18:14:27 ingo Exp $

HISTORY:      07-07-2008 Diepholz
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [xyzdata](#) XZYDATA X-Y-Z data object class constructor.

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      update_struct
0005 %
0006 % DESCRIPTION:   UPDATE_STRUCT update the input structure to the current
0007 %               ltpda version
0008 %
0009 % CALL:         obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:      $Id: update_struct.m,v 1.1 2008/07/25 18:14:27 ingo Exp $
0012 %
0013 % HISTORY:      07-07-2008 Diepholz
0014 %               Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function varargout = update_struct(varargin)
0019
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028
```

```

0029 %%%%%%%%%% Update version '1.0 -> 1.1' %%%%%%%%%%
0030
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035         obj_struct = rmfield(obj_struct, 'name');
0036         obj_struct = rmfield(obj_struct, 'created');
0037     catch ME
0038         disp(varargin{1});
0039         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (fsdata) above is not
from version 1.0')));
0040     end
0041
0042     % Set the struct_ver to '1.1'
0043     struct_ver = '1.1';
0044
0045 end
0046
0047 %%%%%%%%%% Update version '1.1' ->'1.2' %%%%%%%%%%
0048
0049 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0050     strcmp(struct_ver, '1.1')
0051
0052     % Set the struct_ver to '1.2'
0053     struct_ver = '1.1';
0054
0055 end
0056
0057 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0058
0059 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0060     strcmp(struct_ver, '1.2')
0061
0062     % Set the struct_ver to '1.3'
0063     struct_ver = '1.3';
0064
0065 end
0066
0067 varargout{1} = obj_struct;
0068

```

data3D

PURPOSE ^

DATA3D is the abstract base class for 3-dimensional data objects.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
DATA3D is the abstract base class for 3-dimensional data objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DATA3D is the base class for 3-dimensional data objects. This is
              an abstract class.

SUPER CLASSES: ltpda_data < ltpda_nuo < ltpda_obj

SUB CLASSES:   xydata, tsdata, fsdata, data3D

PROPERTIES:

    Inherit Properties (read only)
        version - cvs-version string.

    Protected Properties (read only)
        xunits - units of the y-axis
        yunits - units of the y-axis
        x      - data values of the x-axis
        y      - data values of the y-axis

DATA3D METHODS:

    Public Methods
        char          - returns one character string which represents the object
        setZunits     - set the property 'xunits'
        setZ          - set the property 'z'
        setXY         - set the property 'x' and 'y'
        setXYZ        - set the property 'x' and 'y' and 'z'
        getZ          - get the property 'z'
        applymethod   - applys the given method to the input 3D data
        applyoperator - applys the given operator to the two input data objects

VERSION:   $Id: data3D.m,v 1.8 2008/09/08 08:29:13 hewitson Exp $

HISTORY:   09-06-2008 Hewitson
            Creation.

SEE ALSO:  ltpda_data, fsdata, tsdata, xydata, data3D
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [data3D](#) DATA3D is the abstract base class for 3-dimensional data objects.
-  [getZ](#) GETZ Get the property 'z'.
-  [setZ](#) SETZ Set the property 'z'.
-  [setZunits](#) SETZUNITS Set the property 'zunits'.

This function is called by:

-  [data3D](#) DATA3D is the abstract base class for 3-dimensional data objects.

SUBFUNCTIONS ^

- function obj = set.zunits(obj, val)
- function obj = set.z(obj, val)
- function obj = data3D(varargin)
- function varargout = applymethod(varargin), error(
- function varargout = applyoperator(varargin), error(
- function ii = getInfo(varargin)
- function out = VEROUT()
- function out = SETS()
- function out = getDefaultPlist()

SOURCE CODE ^

```
0001 % DATA3D is the abstract base class for 3-dimensional data objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DATA3D is the base class for 3-dimensional data objects. This is
0005 %               an abstract class.
0006 %
0007 % SUPER CLASSES: ltpda_data < ltpda_nuo < ltpda_obj
0008 %
0009 % SUB CLASSES:   xydata, tsdata, fsdata, data3D
0010 %
0011 % PROPERTIES:
0012 %
0013 %     Inherit Properties (read only)
0014 %         version - cvs-version string.
0015 %
0016 %     Protected Properties (read only)
0017 %         xunits - units of the y-axis
0018 %         yunits - units of the y-axis
0019 %         x      - data values of the x-axis
0020 %         y      - data values of the y-axis
0021 %
0022 % DATA3D METHODS:
0023 %
0024 %     Public Methods
0025 %         char - returns one character string which represents the object
0026 %         setZunits - set the property 'xunits'
0027 %         setZ      - set the property 'z'
0028 %         setXY     - set the property 'x' and 'y'
0029 %         setXYZ    - set the property 'x' and 'y' and 'z'
0030 %         getZ      - get the property 'z'
0031 %         applymethod - applies the given method to the input 3D data
0032 %         applyoperator - applies the given operator to the two input data objects
0033 %
0034 % VERSION:   $Id: data3D.m,v 1.8 2008/09/08 08:29:13 hewitson Exp $
0035 %
0036 % HISTORY:   09-06-2008 Hewitson
0037 %             Creation.
0038 %
0039 % SEE ALSO:  ltpda_data, fsdata, tsdata, xydata, data3D
0040 %
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042
0043 classdef data3D < data2D
0044
0045
0046 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0047 %                               Property definition                               %
0048 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0049
0050 %----- Public (read/write) Properties -----
0051 properties
0052 end
0053
0054 %----- Protected read-only Properties -----
0055 properties
0056     zunits = unit;
0057     z      = [];
0058 end
0059
0060 %----- Private Properties -----
0061 properties (GetAccess = protected, SetAccess = protected)
0062 end
```

```

0063
0064 %----- Abstract Properties -----
0065 properties (Abstract = true, SetAccess = protected)
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %                                     Check property setting
0070 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0071
0072 methods
0073 %--- Zunits
0074 function obj = set.zunits(obj, val)
0075     if ~ischar(val) && ~isa(val, 'unit')
0076         error('### The value for the property ''zunits'' must be a unit-object');
0077     end
0078     if ischar(val)
0079         obj.zunits = unit(val);
0080     elseif isa(val, 'unit')
0081         obj.zunits = val;
0082     else
0083         error('### The zunits value must be a unit or a string');
0084     end
0085 end
0086 %--- Z
0087 function obj = set.z(obj, val)
0088     if ~isnumeric(val) || ndims(val) ~= 2 || isvector(val)
0089         error('### The value for the property ''z'' must be a numeric matrix');
0090     end
0091     obj.z = val;
0092 end
0093 end
0094
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096 %                                     Constructor
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098
0099 methods
0100     function obj = data3D(varargin)
0101
0102         %% Call superclass
0103         obj = obj@data2D(varargin{:});
0104
0105         %%%%%%%%% Set default values %%%%%%%%%
0106
0107         if nargin == 1
0108             if isstruct(varargin{1})
0109                 %%%%%%%%% obj = data3D(struct) %%%%%%%%%
0110
0111                 %% Set properties which are declared in this class
0112                 data3D_struct = varargin{1};
0113
0114                 obj.zunits = data3D_struct.zunits;
0115                 obj.z = data3D_struct.z;
0116             end
0117         end
0118     end % End constructor
0119
0120 end
0121
0122 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0123 %                                     Methods (public)
0124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0125
0126 methods
0127     varargout = setZunits(varargin)
0128     varargout = setZ(varargin)
0129     varargout = getZ(varargin)
0130
0131     function varargout = applymethod(varargin), error('### code me up because we
Inherit this fcn. '); end
0132     function varargout = applyoperator(varargin), error('### code me up because we
Inherit this fcn. '); end
0133 end
0134
0135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0136 %                                     Methods (protected)
0137 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0138
0139 methods (Access = protected)
0140 end
0141
0142 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0143 %                                     Methods (public)
0144 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0145
0146 methods

```



```
0147 end
0148
0149 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0150 %                                Methods (static)                                %
0151 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0152
0153 methods (Static)
0154
0155     function ii = getInfo(varargin)
0156         ii = utils.helper.generic_getInfo(varargin{:}, 'data3D');
0157     end
0158
0159     function out = VEROUT()
0160         out = '$Id: data3D.m,v 1.8 2008/09/08 08:29:13 hewitson Exp $';
0161     end
0162
0163     function out = SETS()
0164         out = {};
0165     end
0166
0167     function out = getDefaultPlist()
0168         out = [];
0169     end
0170
0171 end
0172
0173 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0174 %                                Methods (abstract)                                %
0175 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0176
0177 methods (Static)
0178 end
0179
0180 end
0181
```

Contents

PURPOSE ^

%% class: ao %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ao %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help ao/abs">ao/abs</a>                - overloads the Absolute value
method for Analysis objects.
    <a href="matlab:help ao/acos">ao/acos</a>                - overloads the acos method for
Analysis objects.
    <a href="matlab:help ao/ao">ao/ao</a>                    - analysis object class
constructor.
    <a href="matlab:help ao/ao2m">ao/ao2m</a>                - converts an analysis object to
an '.m' file based on the history.
    <a href="matlab:help ao/applymethod">ao/applymethod</a>    - to the analysis object
    <a href="matlab:help ao/applyoperator">ao/applyoperator</a> - to the analysis
object
    <a href="matlab:help ao/asin">ao/asin</a>                - overloads the asin method for
Analysis objects.
    <a href="matlab:help ao/atan">ao/atan</a>                - overloads the atan method for
Analysis objects.
    <a href="matlab:help ao/atan2">ao/atan2</a>                - overloads the atan2 operator
for Analysis objects. Four quadrant inverse tangent.
    <a href="matlab:help ao/attachm">ao/attachm</a>            - attach an m file to the
analysis object.
    <a href="matlab:help ao/attachmdl">ao/attachmdl</a>        - attach an mdl file to the
analysis object.
    <a href="matlab:help ao/cat">ao/cat</a>                  - concatenate AOs into a vector.
    <a href="matlab:help ao/char">ao/char</a>                - overloads char() function for
analysis objects.
    <a href="matlab:help ao/cohere">ao/cohere</a>              - makes coherence estimates of
the time-series objects
    <a href="matlab:help ao/complex">ao/complex</a>            - overloads the complex
operator for Analysis objects.
    <a href="matlab:help ao/compute">ao/compute</a>            - performs the given
operations on the input AOs.
    <a href="matlab:help ao/computeDFT">ao/computeDFT</a>        - Computes DFT using FFT
or Goertzel
    <a href="matlab:help ao/computeperiodogram">ao/computeperiodogram</a> - Periodogram
spectral estimation.
    <a href="matlab:help ao/conj">ao/conj</a>                  - overloads the conjugate
operator for Analysis objects.
    <a href="matlab:help ao/consolidate">ao/consolidate</a>      - resamples all input AOs
onto the same time grid.
    <a href="matlab:help ao/conv">ao/conv</a>                  - vector convolution.
    <a href="matlab:help ao/conv_noisegen">ao/conv_noisegen</a>    - calls the matlab
function conv.m to convolute poles and zeros from a given pzmodel
    <a href="matlab:help ao/copy">ao/copy</a>                  - makes a (deep) copy of the
input AOs.
    <a href="matlab:help ao/cos">ao/cos</a>                    - overloads the cos operator for
Analysis objects. Cosine of argument in radians.
    <a href="matlab:help ao/cpsd">ao/cpsd</a>                  - makes cross-spectral density
estimates of the time-series objects.
    <a href="matlab:help ao/ctranspose">ao/ctranspose</a>        - overloads the ' operator
for Analysis Objects.
    <a href="matlab:help ao/delay">ao/delay</a>                - delays a time-series using
various methods.
    <a href="matlab:help ao/demux">ao/demux</a>                - splits the input vector of
AOs into a number of output AOs.
    <a href="matlab:help ao/det">ao/det</a>                    - overloads the determinant
function for Analysis objects.
    <a href="matlab:help ao/detrend">ao/detrend</a>            - detrends the input analysis
object using a polynomial of degree N.
    <a href="matlab:help ao/dft">ao/dft</a>                    - computes the DFT of the input
time-series at the requested frequencies.
    <a href="matlab:help ao/diag">ao/diag</a>                  - overloads the diagonal
```

operator for Analysis Objects.

- [ao/diff](matlab:help ao/diff) - differentiates the data in AO.
- [ao/display](matlab:help ao/display) - implement terminal display for analysis object.
- [ao/dopplercorr](matlab:help ao/dopplercorr) - F00 description for function 'foo' in one line. Necessary for lookfor functionality.
- [ao/downsample](matlab:help ao/downsample) - AOs containing time-series data.
- [ao/dropduplicates](matlab:help ao/dropduplicates) - drops all duplicate samples in time-series AOs.
- [ao/dsmean](matlab:help ao/dsmean) - performs a simple downsampling by taking the mean of every N samples.
- [ao/eig](matlab:help ao/eig) - overloads the determinant function for Analysis objects.
- [ao/exp](matlab:help ao/exp) - overloads the exp operator for Analysis objects. Exponential.
- [ao/export](matlab:help ao/export) - export an analysis object to a text file.
- [ao/extractm](matlab:help ao/extractm) - extracts an m-file from an analysis object and saves it to disk.
- [ao/extractmdl](matlab:help ao/extractmdl) - extracts an mdl file from an analysis object and saves it to disk.
- [ao/fft](matlab:help ao/fft) - overloads the fft method for Analysis objects.
- [ao/filter](matlab:help ao/filter) - overrides the filter function for analysis objects.
- [ao/filtfilt](matlab:help ao/filtfilt) - overrides the filtfilt function for analysis objects.
- [ao/find](matlab:help ao/find) - particular samples that satisfy the input query and return a new AO.
- [ao/findFsMax](matlab:help ao/findFsMax) - Returns the max Fs of a set of AOs
- [ao/findShortestVector](matlab:help ao/findShortestVector) - Returns the length of the shortest vector in samples
- [ao/firwhiten](matlab:help ao/firwhiten) - whitens the input time-series by building an FIR whitening filter.
- [ao/fixfs](matlab:help ao/fixfs) - resamples the input time-series to have a fixed sample rate.
- [ao/fngen](matlab:help ao/fngen) - creates an arbitrarily long time-series based on the input PSD.
- [ao/fq2fac](matlab:help ao/fq2fac) - is a private function and is called by ngconv.m which can be found in the
- [ao/fromDataInMAT](matlab:help ao/fromDataInMAT) - Convert a saved data-array into an AO with a tsdata-object
- [ao/fromDatafile](matlab:help ao/fromDatafile) - Construct an ao from filename AND parameter list
- [ao/fromFSfcn](matlab:help ao/fromFSfcn) - Construct an ao from a fs-function string
- [ao/fromFcn](matlab:help ao/fromFcn) - Construct an ao from a function string
- [ao/fromPolyval](matlab:help ao/fromPolyval) - Construct a time-series ao from polynomial coefficients
- [ao/fromPzmodel](matlab:help ao/fromPzmodel) - Construct a time-series ao from polynomial coefficients
- [ao/fromSpecWin](matlab:help ao/fromSpecWin) - Construct an ao from a Spectral window
- [ao/fromTSfcn](matlab:help ao/fromTSfcn) - Construct an ao from a ts-function string
- [ao/fromVals](matlab:help ao/fromVals) - Construct an ao from a value set
- [ao/fromWaveform](matlab:help ao/fromWaveform) - Construct an ao from a waveform
- [ao/fs](matlab:help ao/fs) - Get the data property 'fs'.
- [ao/gapfilling](matlab:help ao/gapfilling) - fills possible gaps in data.
- [ao/ge](matlab:help ao/ge) - overloads >= operator for analysis objects. Compare the y-axis values.
- [ao/gt](matlab:help ao/gt) - overloads > operator for analysis objects. Compare the y-axis values.
- [ao/hist](matlab:help ao/hist) - overloads the histogram function (hist) of MATLAB for Analysis Objects.
- [ao/ifft](matlab:help ao/ifft) - overloads the ifft operator for Analysis objects.
- [ao/imag](matlab:help ao/imag) - overloads the imaginary operator for Analysis objects.
- [ao/index](matlab:help ao/index) - index into an AO array or matrix. This properly captures the history.
- [ao/interp](matlab:help ao/interp) - interpolate the values in the input AO(s) at new values.
- [ao/interpmissing](matlab:help ao/interpmissing) - interpolate missing samples in a time-series.
- [ao/inv](matlab:help ao/inv) - overloads the inverse function for Analysis Objects.
- [ao/iplot](matlab:help ao/iplot) - provides an intelligent plotting tool for LTPDA.
- [ao/join](matlab:help ao/join) - multiple AOs into a single AO.

ao/lcohere	- implement coherence
estimation computed on a log frequency axis.	
ao/lcpsd	- implement cross-power-
spectral density estimation computed on a log frequency axis.	
ao/le	- overloads <= operator for
analysis objects. Compare the y-axis values.	
ao/len	- overloads the length operator
for Analysis objects. Length of the data samples.	
ao/lincom	- (No help available)
ao/linedetect	- find spectral lines in
the ao/fsdata objects.	
ao/ln	- overloads the log operator for
Analysis objects. Natural logarithm.	
ao/log	- overloads the log operator for
Analysis objects. Natural logarithm.	
ao/log10	- overloads the log10 operator
for Analysis objects. Common (base 10) logarithm.	
ao/lpsd	- implement the LPSD algorithm
for analysis objects.	
ao/lt	- overloads < operator for
analysis objects. Compare the y-axis values.	
ao/ltf_plan	- computes all input values
needed for the LPSD and LTFE algorithms.	
ao/ltfe	- implement transfer-function
estimation computed on a log frequency axis.	
ao/lxspec	- performs log-scale cross-
spectral analysis of various forms.	
ao/max	- computes the maximum value of
the data in the A0.	
ao/mchol	- (No help available)
ao/md5	- computes an MD5 checksum from
an analysis objects.	
ao/mdc1_ifo2acc_fd	- MDC1_IFO2ACC_FS
calculates the external acceleration in the frequency-domain.	
ao/mdc1_ifo2acc_inloop	- calculates the
inloop acceleration in the time-domain.	
ao/mdc1_ifo2control	- converts the input
time-series to control forces.	
ao/mdc1_input_noises	- %
LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.	
ao/mdc1_x2acc	- converts the input time-
series to acceleration with a time-domain filter	
ao/mean	- computes the mean value of
the data in the A0.	
ao/median	- computes the median value of
the data in the A0.	
ao/min	- computes the minimum value of
the data in the A0.	
ao/minus	- implements minus operator for
analysis objects.	
ao/mlpsd_m	- m-file only version of the
LPSD algorithm	
ao/mlpsd_mex	- calls the ltpda_dft.mex
to compute the DFT part of the LPSD	
ao/mltfe	- compute log-frequency space
TF	
ao/mode	- computes the modal value of
the data in the A0.	
ao/mpower	- implements mpower operator
for analysis objects.	
ao/mrdivide	- implements mrdivide
operator for analysis objects.	
ao/mtimes	- implements mtimes operator
for analysis objects.	
ao/ngconv	- is called by the function
LTPDA_NOISEGEN	
ao/nginit	- is called by the function
LTPDA_NOISEGEN	
ao/ngprop	- is called by the function
LTPDA_NOISEGEN	
ao/ngsetup	- is called by the function
LTPDA_NOISEGEN	
ao/ngsetup_vpa	- % ALGONAME = mfilename;
ao/norm	- overloads the norm operator
for Analysis Objects.	
ao/phase	- overloads the ltpda_phase
operator for Analysis objects.	
ao/plot	- a simple plot of analysis
objects.	
ao/plus	- implements addition operator
for analysis objects.	
ao/polyfit	- overloads polyfit()
function of MATLAB for Analysis Objects.	
ao/power	- implements power operator for
analysis objects.	
ao/psd	- makes power spectral density

estimates of the time-series objects	
ao/pwelch	- makes power spectral density
estimates of the time-series objects	
ao/rdivide	- implements rdivide operator
for analysis objects.	
ao/real	- overloads the real operator
for Analysis objects.	
ao/resample	- overloads resample
function for AOs.	
ao/rms	- Calculate RMS deviation from spectrum
ao/search	- selects AOs that match the given name.
ao/select	- select particular samples
from the input AOs and return new AOs with only those samples.	
ao/setDescription	- sets the 'description' property of the ao.
ao/setFs	- sets the 'fs' property of the ao.
ao/setT0	- sets the 't0' property of the ao.
ao/setX	- sets the 'x' property of the ao.
ao/setXY	- sets the 'xy' property of the ao.
ao/setXunits	- sets the 'xunits' property of the ao.
ao/setY	- sets the 'y' property of the ao.
ao/setYunits	- sets the 'yunits' property of the ao.
ao/setZ	- sets the 'z' property of the ao.
ao/sign	- overloads the sign operator
for Analysis objects.%	
ao/sin	- overloads the sin method for Analysis objects.
ao/smooth	- smooths a given series of data points using the specified method.
ao/sort	- the values in the AO.
ao/spectrogram	- computes a spectrogram of the given ao/tsdata.
ao/spikecleaning	- detects and corrects possible spikes in analysis objects
ao/split	- split an analysis object into the specified segments.
ao/sqrt	- computes the square root of the data in the AO.
ao/std	- computes the standard deviation of the data in the AO.
ao/string	- writes a command string that can be used to recreate the input Analysis object(s).
ao/sum	- computes the sum of the data in the AO.
ao/sg	- overloads the determinant function for Analysis objects.
ao/t0	- Get the data property 't0'.
ao/tan	- overloads the tan method for Analysis objects.
ao/tfe	- makes transfer function estimates of the time-series objects.
ao/timedomainfit	- uses lsconv to fit a set of time-series AOs to a target time-series AO.
ao/times	- implements times operator for analysis objects.
ao/timeshift	- for AO/tsdata objects, shifts the time axis such that x(1) = 0.
ao/transpose	- overloads the .' operator for Analysis Objects.
ao/uminus	- overloads the uminus operator for Analysis objects.
ao/update_struct	- update the input structure to the current ltpda version
ao/upsample	- overloads upsample function for AOs.
ao/validate	- checks that the input Analysis Object is reproducible and valid.
ao/var	- computes the variance of the data in the AO.
ao/welch	- Welch spectral estimation method.
ao/welchparse	- Parser for the PWELCH & SPECTROGRAM functions
ao/welchscale	- scales the output of welch to be in the required units
ao/x	- Get the data property 'x'.

ao/xcorr	- makes cross-correlation estimates of the time-series
ao/xspec	- performs cross-spectral analysis of various forms.
ao/xunits	- Get the data property 'xunits'.
ao/y	- Get the data property 'y'.
ao/yunits	- Get the data property 'yunits'.
ao/zeropad	- zero pads the input data series.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ao2m

PURPOSE ^

AO2M converts an analysis object to an '.m' file based on the history.

SYNOPSIS ^

function varargout = ao2m(varargin)

DESCRIPTION ^

AO2M converts an analysis object to an '.m' file based on the history.
%%%

DESCRIPTION: AO2M converts an analysis object to an '.m' file based on the history contained in the analysis object.

CALL: ao2m(a, 'filename');
 ao2m(a, plist);

PARAMETERS: 'filename' - the filename to write to

VERSION: \$Id: ao2m.m,v 1.26 2008/09/08 09:00:39 hewitson Exp \$

HISTORY: 08-02-07 M Hewitson
 Creation

%%%

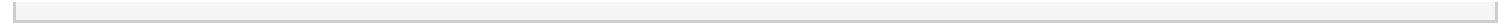
CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % AO2M converts an analysis object to an '.m' file based on the history.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: AO2M converts an analysis object to an '.m' file based on the
0005 %             history contained in the analysis object.
0006 %
0007 % CALL:      ao2m(a, 'filename');
0008 %           ao2m(a, plist);
0009 %
0010 % PARAMETERS: 'filename' - the filename to write to
0011 %
0012 % VERSION: $Id: ao2m.m,v 1.26 2008/09/08 09:00:39 hewitson Exp $
0013 %
0014 % HISTORY: 08-02-07 M Hewitson
0015 %           Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018
0019 function varargout = ao2m(varargin)
0020
0021
0022     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/type now.');
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

Contents

PURPOSE ^

%% class: cdata %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: cdata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help cdata/applymethod">cdata/applymethod</a>    -  applies the given method
to the input 2D data.
    <a href="matlab:help cdata/applyoperator">cdata/applyoperator</a> -  applies the given
operator to the two input data objects.
    <a href="matlab:help cdata/cdata">cdata/cdata</a>                 -  is the constant data class.
    <a href="matlab:help cdata/char">cdata/char</a>                   -  convert a cdata-object into a
string.
    <a href="matlab:help cdata/copy">cdata/copy</a>                   -  Make copy of cdata objects
depending of the second input
    <a href="matlab:help cdata/display">cdata/display</a>             -  implement terminal display
for cdata object.
    <a href="matlab:help cdata/getY">cdata/getY</a>                   -  Get the property 'y'.
    <a href="matlab:help cdata/setY">cdata/setY</a>                   -  Set the property 'y'.
    <a href="matlab:help cdata/setYunits">cdata/setYunits</a>        -  Set the property
'yunits'.
    <a href="matlab:help cdata/update_struct">cdata/update_struct</a> -  update the input
structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: data2D %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: data2D %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help data2D/applymethod">data2D/applymethod</a>    -  applies the given
method to the input 2D data.
    <a href="matlab:help data2D/applyoperator">data2D/applyoperator</a> -  applies the given
operator to the two input data objects.
    <a href="matlab:help data2D/char">data2D/char</a>                    -  convert a ltpda_data-object
into a string.
    <a href="matlab:help data2D/data2D">data2D/data2D</a>                -  is the abstract base class
for 2-dimensional data objects.
    <a href="matlab:help data2D/getX">data2D/getX</a>                    -  Get the property 'x'.
    <a href="matlab:help data2D/getY">data2D/getY</a>                    -  Get the property 'y'.
    <a href="matlab:help data2D/setX">data2D/setX</a>                    -  Set the property 'x'.
    <a href="matlab:help data2D/setXY">data2D/setXY</a>                  -  Set the property 'xy'.
    <a href="matlab:help data2D/setXunits">data2D/setXunits</a>          -  Set the property
'xunits'.
    <a href="matlab:help data2D/setY">data2D/setY</a>                    -  Set the property 'y'.
    <a href="matlab:help data2D/setYunits">data2D/setYunits</a>          -  Set the property
'yunits'.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE [^]

%% class: data3D %%%

SYNOPSIS [^]

This is a script file.

DESCRIPTION [^]

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: data3D %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help data3D/data3D">data3D/data3D</a>      - is the abstract base class for
3-dimensional data objects.
    <a href="matlab:help data3D/getZ">data3D/getZ</a>           - Get the property 'z'.
    <a href="matlab:help data3D/setZ">data3D/setZ</a>           - Set the property 'z'.
    <a href="matlab:help data3D/setZunits">data3D/setZunits</a> - Set the property 'zunits'.
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: fsdata %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: fsdata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help fsdata/copy">fsdata/copy</a>           - Make copy of fsdata objects
depending of the second input
    <a href="matlab:help fsdata/display">fsdata/display</a>     - implement terminal
display for fsdata object.
    <a href="matlab:help fsdata/fsdata">fsdata/fsdata</a>       - frequency-series object
class constructor.
    <a href="matlab:help fsdata/getFfromYFs">fsdata/getFfromYFs</a> - GETDSFROMYFS grows an
evenly spaced frequency vector of N points for
    <a href="matlab:help fsdata/setEnbw">fsdata/setEnbw</a>      - Set the property 'enbw'.
    <a href="matlab:help fsdata/setFs">fsdata/setFs</a>         - Set the property 'fs'.
    <a href="matlab:help fsdata/setNavs">fsdata/setNavs</a>     - Set the property 'navs'.
    <a href="matlab:help fsdata/setT0">fsdata/setT0</a>         - Set the property 't0'.
    <a href="matlab:help fsdata/update_struct">fsdata/update_struct</a> - update the input
structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: history %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: history %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help history/char">history/char</a>           - convert a param object
into a string.
    <a href="matlab:help history/copy">history/copy</a>           - Make copy of history
objects depending of the second input
    <a href="matlab:help history/display">history/display</a>     - implement terminal
display for history object.
    <a href="matlab:help history/getNodes">history/getNodes</a>   - converts a history
object to a nodes structure suitable for plotting as a tree.
    <a href="matlab:help history/getNodes_plot">history/getNodes_plot</a> - converts a
history object to a nodes structure suitable for plotting as a tree.
    <a href="matlab:help history/hist2dot">history/hist2dot</a>   - converts a history
object to a 'DOT' file suitable for processing with graphviz
    <a href="matlab:help history/hist2m">history/hist2m</a>       - writes a new m-file that
reproduces the analysis described in the history object.
    <a href="matlab:help history/history">history/history</a>     - History object class
constructor.
    <a href="matlab:help history/plot">history/plot</a>           - plots a history object as
a tree diagram.
    <a href="matlab:help history/string">history/string</a>       - writes a command string
that can be used to recreate the input history object.
    <a href="matlab:help history/update_struct">history/update_struct</a> - update the input
structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: ltpda_data %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%% class: ltpda_data %%%

ltpda_data/ltpda_data - is the abstract
base class for ltpda data objects.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpda_data

PURPOSE ^

LTPDA_DATA is the abstract base class for ltpda data objects.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_DATA is the abstract base class for ltpda data objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION:  LTPDA_DATA is the base class for ltpda data objects.
               This is an abstract class.

SUPER CLASSES: ltpda_nuo < ltpda_obj

SUB CLASSES:   data2D

PROPERTIES:

LTPDA_data METHODS:

M-FILE INFO:  The following call returns an minfo object that contains
               information about the ltpda_data constructor:
               >> info = ltpda_data.getInfo
               or  >> info = ltpda_data.getInfo('ltpda_data')

VERSION:      $Id: ltpda_data.m,v 1.10 2008/09/03 16:32:48 hewitson Exp $

HISTORY:      09-06-2008 Hewitson
               Creation.

SEE ALSO:     data2D, ltpda_nuo, ltpda_obj, cdata
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpda_data](#) LTPDA_DATA is the abstract base class for ltpda data objects.

This function is called by:

 [ltpda_data](#) LTPDA_DATA is the abstract base class for ltpda data objects.

SUBFUNCTIONS ^

-  [function obj = ltpda_data\(varargin\)](#)
-  [function ii = getInfo\(varargin\)](#)
-  [function out = VEROUT\(\)](#)
-  [function out = SETS\(\)](#)
-  [function out = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % LTPDA DATA is the abstract base class for ltpda data objects.
```

```
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION:    LTPDA_DATA is the base class for ltpda data objects.
0005 %                This is an abstract class.
0006 %
0007 % SUPER CLASSES: ltpda_nuo < ltpda_obj
0008 %
0009 % SUB CLASSES:   data2D
0010 %
0011 % PROPERTIES:
0012 %
0013 % LTPDA_data METHODS:
0014 %
0015 % M-FILE INFO: The following call returns an minfo object that contains
0016 %              information about the ltpda_data constructor:
0017 %              >> info = ltpda_data.getInfo
0018 %              or   >> info = ltpda_data.getInfo('ltpda_data')
0019 %
0020 % VERSION:      $Id: ltpda_data.m,v 1.10 2008/09/03 16:32:48 hewitson Exp $
0021 %
0022 % HISTORY:      09-06-2008 Hewitson
0023 %                Creation.
0024 %
0025 % SEE ALSO:     data2D, ltpda_nuo, ltpda_obj, cdata
0026 %
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029 classdef ltpda_data < ltpda_nuo
0030
0031     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0032     %                                Property definition                                %
0033     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0034
0035     %----- Public (read/write) Properties -----
0036     properties
0037     end
0038
0039     %----- Protected read-only Properties -----
0040     properties (SetAccess = protected)
0041     end
0042
0043     %----- Private Properties -----
0044     properties (GetAccess = protected, SetAccess = protected)
0045     end
0046
0047     %----- Abstract Properties -----
0048     properties (Abstract = true, SetAccess = protected)
0049     end
0050
0051     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0052     %                                Check property setting                                %
0053     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0054
0055     methods
0056     end
0057
0058     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0059     %                                Constructor                                %
0060     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0061
0062     methods
0063
0064         function obj = ltpda_data(varargin)
0065
0066             %% Call superclass
0067             obj = obj@ltpda_nuo(varargin{:});
0068
0069             %%%%%%%%% Set default values %%%%%%%%%
0070         end
0071
0072     end
0073
0074     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0075     %                                Methods (public)                                %
0076     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0077
0078     methods
0079     end
0080
0081     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0082     %                                Methods (protected)                                %
0083     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0084
0085     methods (Access = protected)
0086     end
0087
```

```
0088 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0089 %                                Methods (private)                                %
0090 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0091
0092 methods (Access = private)
0093 end
0094
0095 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0096 %                                Methods (static)                                %
0097 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0098
0099 methods (Static)
0100
0101     function ii = getInfo(varargin)
0102         ii = utils.helper.generic_getInfo(varargin{:}, 'ltpda_data');
0103     end
0104
0105     function out = VEROUT()
0106         out = '$Id: ltpda_data.m,v 1.10 2008/09/03 16:32:48 hewitson Exp $';
0107     end
0108
0109     function out = SETS()
0110         out = {};
0111     end
0112
0113     function out = getDefaultPlist()
0114         out = [];
0115     end
0116
0117 end
0118
0119 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0120 %                                Methods (abstract)                                %
0121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0122
0123 methods (Abstract)
0124 end
0125
0126 end
0127
```

Contents

PURPOSE ^

%% class: ltpda_filter %%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      class: ltpda_filter      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ltpda_filter/ltpda_filter">ltpda_filter/ltpda_filter</a> - is the
abstract base class for ltpda filter objects.
    <a href="matlab:help ltpda_filter/setA">ltpda_filter/setA</a>                - Set the property
'a'
    <a href="matlab:help ltpda_filter/setFs">ltpda_filter/setFs</a>              - Set the property
'fs'
    <a href="matlab:help ltpda_filter/setHistout">ltpda_filter/setHistout</a>    - Set the
property 'histout'
    <a href="matlab:help ltpda_filter/setInfile">ltpda_filter/setInfile</a>    - Set the
property 'infile'
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

setA

PURPOSE ^

SETA Set the property 'a'

SYNOPSIS ^

function obj = setA(obj, val)

DESCRIPTION ^

SETA Set the property 'a'

DESCRIPTION: SETA Set the property 'a'

CALL: obj = obj.setA([0.24 0.24]);
obj = setA(obj, [0.24 0.24]);

INPUTS: obj - is a ltpda_filter object

VERSION: \$Id: setA.m,v 1.1 2008/07/18 11:13:34 ingo Exp \$

HISTORY: 16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_filter](#) LTPDA_FILTER is the abstract base class for ltpda filter objects.

SOURCE CODE ^

```
0001 % SETA Set the property 'a'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETA Set the property 'a'
0005 %
0006 % CALL:      obj = obj.setA([0.24 0.24]);
0007 %           obj = setA(obj, [0.24 0.24]);
0008 %
0009 % INPUTS:    obj - is a ltpda_filter object
0010 %
0011 % VERSION:   $Id: setA.m,v 1.1 2008/07/18 11:13:34 ingo Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function obj = setA(obj, val)
0019     %% decide whether we modify the ltpda_filter-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'a'
0023     obj.a = val;
0024 end
0025
```


setFs

PURPOSE ^

SETFS Set the property 'fs'

SYNOPSIS ^

function obj = setFs(obj, val)

DESCRIPTION ^

SETFS Set the property 'fs'

DESCRIPTION: SETFS Set the property 'fs'

CALL:
obj = obj.setFs(12);
obj = setFs(obj, 12);

INPUTS:
obj - is a ltpda_filter object

VERSION:
\$Id: setFs.m,v 1.1 2008/07/18 11:13:34 ingo Exp \$

HISTORY:
16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_filter](#) LTPDA_FILTER is the abstract base class for ltpda filter objects.

SOURCE CODE ^

```
0001 % SETFS Set the property 'fs'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETFS Set the property 'fs'
0005 %
0006 % CALL:      obj = obj.setFs(12);
0007 %           obj = setFs(obj, 12);
0008 %
0009 % INPUTS:    obj - is a ltpda_filter object
0010 %
0011 % VERSION:   $Id: setFs.m,v 1.1 2008/07/18 11:13:34 ingo Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function obj = setFs(obj, val)
0019     %% decide whether we modify the ltpda_filter-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'fs'
0023     obj.fs = val;
0024 end
0025
```


setInfile

PURPOSE ^

SETINFILE Set the property 'infile'

SYNOPSIS ^

function obj = setInfile(obj, val)

DESCRIPTION ^

```
SETINFILE Set the property 'infile'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: SETINFILE Set the property 'infile'

CALL:      obj = obj.setInfile('file');
           obj = setInfile(obj, 'file');

INPUTS:    obj - is a ltpda_filter object

VERSION:    $Id: setInfile.m,v 1.1 2008/07/18 11:13:34 ingo Exp $

HISTORY:    16-07-2008 Diepholz
            Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_filter](#) LTPDA_FILTER is the abstract base class for ltpda filter objects.

SOURCE CODE ^

```
0001 % SETINFILE Set the property 'infile'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETINFILE Set the property 'infile'
0005 %
0006 % CALL:      obj = obj.setInfile('file');
0007 %           obj = setInfile(obj, 'file');
0008 %
0009 % INPUTS:    obj - is a ltpda_filter object
0010 %
0011 % VERSION:    $Id: setInfile.m,v 1.1 2008/07/18 11:13:34 ingo Exp $
0012 %
0013 % HISTORY:    16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function obj = setInfile(obj, val)
0019     %% decide whether we modify the ltpda_filter-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'infile'
0023     obj.infile = val;
0024 end
0025
```


Contents

PURPOSE ^

%% class: ltpda_nuo %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%% class: ltpda_nuo %%%

 [ltpda_nuo/ltpda_nuo](matlab:help ltpda_nuo/ltpda_nuo) - is the abstract ltpda
base class for ltpda non user object classes.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpda_nuo

PURPOSE ^

LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION:  LTPDA_NUO is the ltpda base class for ltpda non user object classes.
               This class is an abstract class and it is not possible to create
               an instance of this class.
               ALL ltpda non user object classes inherit this 'abstract' class.

SUPER CLASSES: ltpda_obj

SUB CLASSES:   specwin, time, provenance, pz, param, history, minfo, ltpda_data

PROPERTIES:

    Inherit Properties (read only)
    version - cvs-version string.

LTPDA_UO Methods:

    Public Methods
    Protected Methjods

M-FILE INFO: The following call returns an minfo object that contains
              information about the ltpda_nuo constructor:
              >> info = ltpda_nuo.getInfo
              or  >> info = ltpda_nuo.getInfo('ltpda_nuo')

VERSION:      $Id: ltpda_nuo.m,v 1.16 2008/09/03 16:32:47 hewitson Exp $

HISTORY:      19-05-2008 Diepholz
               Creation.

SEE ALSO:     ltpda_obj, specwin, time, provenance, pz, param, history, minfo, ltpda_data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpda_nuo](#) LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.

This function is called by:

 [ltpda_nuo](#) LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.

SUBFUNCTIONS ^

-  function obj = ltpda_nuo(varargin)
-  function ii = getInfo(varargin)
-  function out = VEROUT()
-  function out = SETS()

```
function out = getDefaultPlist()
```

SOURCE CODE ^

```
0001 % LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION:    LTPDA_NUO is the ltpda base class for ltpda non user object classes.
0005 %                This class is an abstract class and it is not possible to create
0006 %                an instance of this class.
0007 %                ALL ltpda non user object classes inherit this 'abstract' class.
0008 %
0009 % SUPER CLASSES: ltpda_obj
0010 %
0011 % SUB CLASSES:   specwin, time, provenance, pz, param, history, minfo, ltpda_data
0012 %
0013 % PROPERTIES:
0014 %
0015 %     Inherit Properties (read only)
0016 %     version - cvs-version string.
0017 %
0018 % LTPDA_UO Methods:
0019 %
0020 %     Public Methods
0021 %     Protected Methjods
0022 %
0023 % M-FILE INFO: The following call returns an minfo object that contains
0024 %               information about the ltpda_nuo constructor:
0025 %               >> info = ltpda_nuo.getInfo
0026 %               or  >> info = ltpda_nuo.getInfo('ltpda_nuo')
0027 %
0028 % VERSION:      $Id: ltpda_nuo.m,v 1.16 2008/09/03 16:32:47 hewitson Exp $
0029 %
0030 % HISTORY:      19-05-2008 Diepholz
0031 %               Creation.
0032 %
0033 % SEE ALSO:     ltpda_obj, specwin, time, provenance, pz, param, history, minfo,
0034 %               ltpda_data
0035 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0036
0037 classdef ltpda_nuo < ltpda_obj
0038
0039     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0040     %                               Property definition                               %
0041     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042
0043     %----- Public (read/write) Properties -----
0044     properties
0045     end
0046
0047     %----- Protected read-only Properties -----
0048     properties (SetAccess = protected)
0049     end
0050
0051     %----- Private Properties -----
0052     properties (GetAccess = protected, SetAccess = protected)
0053     end
0054
0055     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0056     %                               Check property setting                               %
0057     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0058
0059     methods
0060     end
0061
0062     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0063     %                               Constructor                               %
0064     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0065
0066     methods
0067         function obj = ltpda_nuo(varargin)
0068             %% Call superclass
0069             obj = obj@ltpda_obj(varargin{:});
0070
0071             %%% Set default values %%%
0072         end
0073     end
0074
0075     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0076     %                               Methods (public)                               %
0077     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
```

```
0079 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0080
0081 methods
0082 end
0083
0084 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0085 %                               Methods (static)                               %
0086 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0087
0088 methods (Static)
0089
0090     function ii = getInfo(varargin)
0091         ii = utils.helper.generic_getInfo(varargin{:}, 'ltpda_nuo');
0092     end
0093
0094     function out = VEROUT()
0095         out = '$Id: ltpda_nuo.m,v 1.16 2008/09/03 16:32:47 hewitson Exp $';
0096     end
0097
0098     function out = SETS()
0099         out = {};
0100     end
0101
0102     function out = getDefaultPlist()
0103         out = [];
0104     end
0105
0106 end
0107
0108 end
0109
```

Contents

PURPOSE ^

%% class: ltpda_obj %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ltpda_obj %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ltpda_obj/eq">ltpda_obj/eq</a>           - overloads the ==
operator for ltpda objects.
    <a href="matlab:help ltpda_obj/isprop">ltpda_obj/isprop</a>   - tests if the given
field is one of the object properties.
    <a href="matlab:help ltpda_obj/ltpda_obj">ltpda_obj/ltpda_obj</a> - is the abstract
ltpda base class.
    <a href="matlab:help ltpda_obj/ne">ltpda_obj/ne</a>           - overloads the ~=
operator for ltpda objects.
    <a href="matlab:help ltpda_obj/prependVersion">ltpda_obj/prependVersion</a> - Prepend the
input-version to the property 'version'
    <a href="matlab:help ltpda_obj/setVersion">ltpda_obj/setVersion</a> - Set the
property 'version'
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

prependVersion

PURPOSE ^

PREPENDVERSION Prepend the input-version to the property 'version'

SYNOPSIS ^

function obj = prependVersion(obj, val)

DESCRIPTION ^

PREPENDVERSION Prepend the input-version to the property 'version'

DESCRIPTION: PREPENDVERSION Prepend the input-version to the property 'version'

CALL: obj = obj.prependVersion('new ver');
obj = prependVersion(obj, 'new ver');

INPUTS: obj - is a general object

VERSION: \$Id: prependVersion.m,v 1.2 2008/07/18 11:13:34 ingo Exp \$

HISTORY: 16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_obj](#) LTPDA_OBJ is the abstract ltpda base class.

SOURCE CODE ^

```
0001 % PREPENDVERSION Prepend the input-version to the property 'version'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: PREPENDVERSION Prepend the input-version to the property 'version'
0005 %
0006 % CALL:      obj = obj.prependVersion('new ver');
0007 %           obj = prependVersion(obj, 'new ver');
0008 %
0009 % INPUTS:    obj - is a general object
0010 %
0011 % VERSION:   $Id: prependVersion.m,v 1.2 2008/07/18 11:13:34 ingo Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function obj = prependVersion(obj, val)
0019     %% decide whether we modify the ltpda_obj-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'version'
0023     obj.version = [val ' --> ' obj.version];
0024 end
```


setVersion

PURPOSE ^

SETVERSION Set the property 'version'

SYNOPSIS ^

function setVersion(obj, val)

DESCRIPTION ^

SETVERSION Set the property 'version'

DESCRIPTION: SETVERSION Set the property 'version'

CALL: obj = obj.setVersion('new ver');
obj = setVersion(obj, 'new ver');

INPUTS: obj - is a general object

VERSION: \$Id: setVersion.m,v 1.7 2008/09/03 16:38:51 hewitson Exp \$

HISTORY: 16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_obj](#) LTPDA_OBJ is the abstract ltpda base class.

SOURCE CODE ^

```
0001 % SETVERSION Set the property 'version'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETVERSION Set the property 'version'
0005 %
0006 % CALL:      obj = obj.setVersion('new ver');
0007 %           obj = setVersion(obj, 'new ver');
0008 %
0009 % INPUTS:    obj - is a general object
0010 %
0011 % VERSION:   $Id: setVersion.m,v 1.7 2008/09/03 16:38:51 hewitson Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function setVersion(obj, val)
0019     %% set 'version'
0020     obj.version = val;
0021 end
```


Contents

PURPOSE ^

%% class: ltpda_uo %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ltpda_uo %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ltpda_uo/ltpda_uo">ltpda_uo/ltpda_uo</a>    - is the abstract ltpda
base class for ltpda user object classes.
    <a href="matlab:help ltpda_uo/retrieve">ltpda_uo/retrieve</a>    - retrieves a collection
of objects from an LTPDA repository.
    <a href="matlab:help ltpda_uo/save">ltpda_uo/save</a>            - overloads save operator for
ltpda objects.
    <a href="matlab:help ltpda_uo/setCreated">ltpda_uo/setCreated</a> - Set the property
'created'
    <a href="matlab:help ltpda_uo/setCreator">ltpda_uo/setCreator</a> - Set the property
'creator'
    <a href="matlab:help ltpda_uo/setName">ltpda_uo/setName</a>      - Set the property 'name'.
    <a href="matlab:help ltpda_uo/submit">ltpda_uo/submit</a>      - submits the given
collection of objects to an LTPDA Repository.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

retrieve

PURPOSE ^

RETRIEVE retrieves a collection of objects from an LTPDA repository.

SYNOPSIS ^

function varargout = retrieve(varargin)

DESCRIPTION ^

```
RETRIEVE retrieves a collection of objects from an LTPDA repository.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: RETRIEVE a collection of objects from an LTPDA repository.

CALL:      objs      = retrieve(conn, obj_id_1, obj_id_2)
           [o1,o2] = retrieve(conn, obj_id_1, obj_id_2)
           objs     = retrieve(conn, 'Collection', coll_id)

PARAMETERS:

INPUTS:
  conn      - database connection object
  obj_id_N  - an object ID
  coll_id   - a collection ID

OUTPUTS:
  objs      - the retrieved object(s) as a cell array.*
  o1,o2,...,oN - returns the first N objects

If more than one object is retrieved and only one output is specified
then the output is a cell array of objects.

If only a single object is requested, it is returned as an object,
not packed in a cell array.

M-FILE INFO: Get information about this methods by calling
              >> ao.getInfo('retrieve')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('retrieve', 'Default')

VERSION:      $Id: retrieve.m,v 1.4 2008/09/04 15:29:30 ingo Exp $

HISTORY:      30-08-07 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```

0001 % RETRIEVE retrieves a collection of objects from an LTPDA repository.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RETRIEVE a collection of objects from an LTPDA repository.
0005 %
0006 % CALL:      objs      = retrieve(conn, obj_id_1, obj_id_2)
0007 %           [o1,o2] = retrieve(conn, obj_id_1, obj_id_2)
0008 %           objs      = retrieve(conn, 'Collection', coll_id)
0009 %
0010 % PARAMETERS:
0011 %
0012 %
0013 % INPUTS:
0014 %         conn      - database connection object
0015 %         obj_id_N  - an object ID
0016 %         coll_id   - a collection ID
0017 %
0018 % OUTPUTS:
0019 %         objs      - the retrieved object(s) as a cell array.*
0020 %         o1,o2,...,oN - returns the first N objects
0021 %
0022 %
0023 % If more than one object is retrieved and only one output is specified
0024 % then the output is a cell array of objects.
0025 %
0026 % If only a single object is requested, it is returned as an object,
0027 % not packed in a cell array.
0028 %
0029 % M-FILE INFO: Get information about this methods by calling
0030 %               >> ao.getInfo('retrieve')
0031 %
0032 %               Get information about a specified set-plist by calling:
0033 %               >> ao.getInfo('retrieve', 'Default')
0034 %
0035 % VERSION:      $Id: retrieve.m,v 1.4 2008/09/04 15:29:30 ingo Exp $
0036 %
0037 % HISTORY:      30-08-07 M Hewitson
0038 %               Creation
0039 %
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041 %
0042 function varargout = retrieve(varargin)
0043 %
0044 % Check if this is a call for parameters
0045 if utils.helper.isinfocall(varargin{:})
0046     varargout{1} = getInfo(varargin{3});
0047     return
0048 end
0049
0050 import utils.const.*
0051 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0052
0053 if nargin == 0
0054     help(mfilename);
0055     error('### Incorrect inputs');
0056 end
0057
0058 objs = [];
0059 conn = varargin{1};
0060 if ~isa(conn, 'database')
0061     error('### the first argument should be a database connection object.');
```

```

0083     error('### Incorrect usage');
0084 end
0085 elseif nargin > 1 && isnumeric(varargin{2})
0086     ids = [varargin{2:end}];
0087 else
0088     error('### Incorrect usage');
0089 end
0090
0091 utils.helper.msg(msg.PROC1, ['retrieving objects [' num2str(ids) ']']);
0092
0093 for j=1:length(ids)
0094     % Get object
0095     xdoc = utils.mysql.getXdoc(conn, ids(j));
0096     if ~isempty(xdoc)
0097         obj = utils.helper.xmlread(xdoc);
0098         if j==1
0099             objs = {obj};
0100         else
0101             objs = [objs {obj}];
0102         end
0103
0104         % make transaction entry
0105         t = time();
0106         tdate = t.format('yyyy-mm-dd HH:MM:SS');
0107         try
0108             message = utils.mysql.insert(conn, ...
0109                 'transactions',...
0110                 'obj_id', ids(j),...
0111                 'user_id', userid,...
0112                 'transdate', tdate,...
0113                 'direction', 'out'...
0114             );
0115             utils.helper.msg(msg.PROC1, 'updated transactions table');
0116         catch
0117             error('### Failed to make entry in transactions table');
0118         end
0119     else
0120         warning('!!! Error retrieving object: %d', ids(j));
0121     end
0122 end
0123
0124 % Set outputs
0125 if nargin == 1
0126     if length(objs) == 1
0127         varargout{1} = objs{1};
0128     else
0129         varargout{1} = objs;
0130     end
0131 else
0132     for k=1:nargout
0133         varargout{k} = objs{k};
0134     end
0135 end
0136 end
0137
0138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0139 % Local Functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0141 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0142 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0143 %
0144 function ii = getInfo(varargin)
0145     if nargin == 1 && strcmpi(varargin{1}, 'None')
0146         sets = {};
0147         pl = [];
0148     else
0149         sets = {'Default'};
0150         pl = getDefaultPlist;
0151     end
0152     % Build info object
0153     ii = minfo(mfilename, 'ltpda_uo', '', utils.const.categories.internal, '$Id:
retrieve.m,v 1.4 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0154 end
0155
0156 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0157 %
0158 function plo = getDefaultPlist()
0159     plo = plist('conn', [], 'ids', []);
0160 end
0161
0162
0163

```

setCreated

PURPOSE ^

SETCREATED Set the property 'created'

SYNOPSIS ^

function obj = setCreated(obj, val)

DESCRIPTION ^

SETCREATED Set the property 'created'

DESCRIPTION: SETCREATED Set the property 'created'

CALL:
obj = obj.setCreated('unknown');
obj = setCreated(obj, 'unknown');

INPUTS:
obj - is a general object

VERSION:
\$Id: setCreated.m,v 1.6 2008/09/03 16:36:23 hewitson Exp \$

HISTORY:
16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

SOURCE CODE ^

```
0001 % SETCREATED Set the property 'created'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETCREATED Set the property 'created'
0005 %
0006 % CALL:      obj = obj.setCreated('unknown');
0007 %            obj = setCreated(obj, 'unknown');
0008 %
0009 % INPUTS:    obj - is a general object
0010 %
0011 % VERSION:   $Id: setCreated.m,v 1.6 2008/09/03 16:36:23 hewitson Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %            Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function obj = setCreated(obj, val)
0019     %% set 'created'
0020     obj.created = val;
0021 end
```


setCreator

PURPOSE ^

SETCREATOR Set the property 'creator'

SYNOPSIS ^

function obj = setCreator(obj, val)

DESCRIPTION ^

SETCREATOR Set the property 'creator'

DESCRIPTION: SETCREATOR Set the property 'creator'

CALL: obj = obj.setCreator('unknown');
obj = setCreator(obj, 'unknown');

INPUTS: obj - is a general object

VERSION: \$Id: setCreator.m,v 1.2 2008/07/18 11:13:34 ingo Exp \$

HISTORY: 16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

SOURCE CODE ^

```
0001 % SETCREATOR Set the property 'creator'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETCREATOR Set the property 'creator'
0005 %
0006 % CALL:      obj = obj.setCreator('unknown');
0007 %           obj = setCreator(obj, 'unknown');
0008 %
0009 % INPUTS:    obj - is a general object
0010 %
0011 % VERSION:   $Id: setCreator.m,v 1.2 2008/07/18 11:13:34 ingo Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function obj = setCreator(obj, val)
0019     %% decide whether we modify the ltpda_uo-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'creator'
0023     obj.creator = val;
0024 end
```


submit

PURPOSE ^

SUBMIT submits the given collection of objects to an LTPDA Repository.

SYNOPSIS ^

function varargout = submit(varargin)

DESCRIPTION ^

```
SUBMIT submits the given collection of objects to an LTPDA Repository.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: Submits the given collection of objects to an LTPDA Repository.
              The input object(s) will be submitted to the LTPDA Repository
              configured in startup.m. If multiple objects are submitted
              together, a corresponding collection entry will be made.

              This is meant to be called by class/submit methods.

CALL:         [ids, cids] = submit(o1, sinfo)
              [ids, cids] = submit(o1,o2, sinfo)
              [ids, cids] = submit([o1 o2], o3, {o4, o5}, sinfo)

INPUTS:       o1,o2,... - input objects to be submitted
              sinfo      - a structure containing the following fields:

              'conn'          - database connection object
              'experiment_title' - a title for the submission (Mandatory, >4
characters)
              'experiment_description' - a description of this submission (Mandatory, >10
characters)
              'analysis_description' - a description of the analysis performed
(Mandatory, >10 characters);
              'quantity'        - the physical quantity represented by the data);
              'keywords'        - a comma-delimited list of keywords);
              'reference_ids'    - a string containing any reference object id
numbers
              'additional_comments' - any additional comments
              'additional_authors' - any additional author names

M-FILE INFO:  Get information about this methods by calling
              >> ao.getInfo('submit')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('submit', 'Default')

VERSION:      $Id: submit.m,v 1.6 2008/09/04 15:29:30 ingo Exp $

HISTORY:      09-05-07 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uo](#) LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

SUBFUNCTIONS ^

 function eid = update_object_tables(varargin)
 function id = getObjID(conn, hash)
 function Cid = getCollectionID(conn, objids)
 function s = csv(x)
 function sinfo = check_sinfo(sinfo)
 function sinfo_description
 function ii = getInfo(varargin)
 function plo = getDefaultPlist()

SOURCE CODE ^

```

0001 % SUBMIT submits the given collection of objects to an LTPDA Repository.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: Submits the given collection of objects to an LTPDA Repository.
0005 %               The input object(s) will be submitted to the LTPDA Repository
0006 %               configured in startup.m. If multiple objects are submitted
0007 %               together, a corresponding collection entry will be made.
0008 %
0009 %               This is meant to be called by class/submit methods.
0010 %
0011 % CALL:         [ids, cids] = submit(o1, sinfo)
0012 %               [ids, cids] = submit(o1,o2, sinfo)
0013 %               [ids, cids] = submit([o1 o2], o3, {o4, o5}, sinfo)
0014 %
0015 % INPUTS:
0016 %               o1,o2,... - input objects to be submitted
0017 %               sinfo    - a structure containing the following fields:
0018 %
0019 %                   'conn'                - database connection object
0020 %                   'experiment_title'    - a title for the submission (Mandatory, >4
characters)
0021 %                   'experiment_description' - a description of this submission
(Mandatory, >10 characters)
0022 %                   'analysis_description' - a description of the analysis performed
(Mandatory, >10 characters);
0023 %                   'quantity'           - the physical quantity represented by the
data);
0024 %                   'keywords'           - a comma-delimited list of keywords);
0025 %                   'reference_ids'       - a string containing any reference object
id numbers
0026 %                   'additional_comments' - any additional comments
0027 %                   'additional_authors'  - any additional author names
0028 %
0029 % M-FILE INFO: Get information about this methods by calling
0030 %               >> ao.getInfo('submit')
0031 %
0032 %               Get information about a specified set-plist by calling:
0033 %               >> ao.getInfo('submit', 'Default')
0034 %
0035 % VERSION:      $Id: submit.m,v 1.6 2008/09/04 15:29:30 ingo Exp $
0036 %
0037 % HISTORY:      09-05-07 M Hewitson
0038 %               Creation
0039 %
0040 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0041
0042 function varargout = submit(varargin)
0043
0044 %% Process inputs
0045 %% Check if this is a call for parameters
0046 if utils.helper.isinfocall(varargin{:})
0047     varargout{1} = getInfo(varargin{3});
0048     return
0049 end
0050
0051 import utils.const.*
0052 utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);
0053
0054 %% Collect all AOs
0055 %   objs = utils.helper.collect_objects(varargin(:), 'ltpda_uo');
0056 [sinfo, invars, objs] = utils.helper.collect_objects(varargin(:), 'struct');
0057
0058 if isempty(objs)
0059     error('### Please input at least one LTPDA user object.');
```

```

0063 end
0064
0065 %% Some setup
0066 % Check the comments
0067 sinfo = check_sinfo(sinfo);
0068
0069 % make copy of connection object
0070 conn = sinfo.conn;
0071 if isempty(conn)
0072     error('### No valid database connect supplied. Aborting submit process.');
```

```

0073 end
0074
0075 %% Process each object and collect id numbers
0076
0077 utils.helper.msg(msg.PROC1, 'submitting objects to repository.');
```

```

0078
0079 try
0080
0081     % Look-up user id
0082     userid = utils.mysql.getUserID(conn);
0083     utils.helper.msg(msg.PROC1, 'got user id %d for user: %s', userid, conn.Username);
0084     if userid < 1 || isnan(userid) || strcmp(userid, 'No Data') || ischar(userid)
0085         error('### Unknown username.');
```

```

0086     end
0087
0088     % The date for the transaction table
0089     t = time();
0090     tdate = format(t, 'yyyy-mm-dd HH:MM:SS');
```

```

0091
0092     % Machine details
0093     prov = provenance();
0094
0095     ids = [];
0096     for kk=1:numel(objs)
0097         % this object
0098         obj = objs{kk};
0099
0100         for jj=1:numel(obj)
0101             utils.helper.msg(msg.PROC1, 'submitting object: %s / %s', class(obj(jj)),
0102 obj(jj).name);
0103
0104             % Object name
0105             name = obj(jj).name;
0106
0107             % Creation time
0108             if isa(obj(jj).created, 'time')
0109                 created = obj(jj).created.format('yyyy-mm-dd HH:MM:SS');
```

```

0110             else
0111                 created = '1970-01-01 00:00:00';
0112             end
0113
0114             % Version
0115             objver = obj(jj).version;
0116
0117             %%%%%%%%%%% Convert object to XML
0118             % make pointer to xml document
0119             xml = com.mathworks.xml.XMLUtils.createDocument('ltpda_object');
```

```

0120             % extract parent node
0121             parent = xml.getDocumentElement;
0122             % add Attribute 'ltpda_version' to the root node
0123             ltpda_version = getappdata(0, 'ltpda_version');
0124             parent.setAttribute('ltpda_version', ltpda_version);
0125             % write obj into xml
0126             utils.helper.xmlwrite(obj(jj), xml, parent, ''); % Save the XML document.
0127             otxt = xmlwrite(xml); % then to string
0128
0129             %%%%%%%%%%% Create MD5 hash of object
0130             md5hash = utils.prog.hash(otxt, 'MD5');
```

```

0131
0132             %%%%%%%%%%% Submit object to objs table
0133             try
0134                 utils.helper.msg(msg.PROC1, 'uploading XML data...\f');
0135                 curs = exec(conn, ['insert into objs(xml,hash) values('' otxt '', ''
0136 char(md5hash) '' );']);
0137                 close(curs);
0138                 utils.helper.msg(msg.PROC1, '\bdone.');
```

```

0139             catch
0140                 utils.helper.msg(msg.PROC1, '### Server returned: %s', curs.Message);
0141                 error('### Submission error');
```

```

0142             end
0143
0144             try
0145                 objid = getObjID(conn, md5hash);
0146                 utils.helper.msg(msg.PROC1, 'submitted object %s with id %d', class(obj(jj)),
0147 objid);
0148             catch

```

```

0146         error('### Failed to get ID of submitted object.');
```

```

0147     end
```

```

0148     %%%%%%%%%%% Make objmeta entry
```

```

0150     % For now reference IDs are stored in a string
```

```

0151     if ischar(sinfo.reference_ids)
```

```

0152         refIds = sinfo.reference_ids;
```

```

0153     else
```

```

0154         refIds = csv(sinfo.reference_ids);
```

```

0155     end
```

```

0156     try
```

```

0157         message = utils.mysql.insert(conn, ...
```

```

0158             'objmeta', ...
```

```

0159             'obj_id', objid, ...
```

```

0160             'obj_type', class(obj(jj)), ...
```

```

0161             'name', name, ...
```

```

0162             'created', created, ...
```

```

0163             'version', objver, ...
```

```

0164             'ip', prov.ip, ...
```

```

0165             'hostname', prov.hostname, ...
```

```

0166             'os', prov.os, ...
```

```

0167             'submitted', datestr(now, 31), ...
```

```

0168             'experiment_title', sinfo.experiment_title, ...
```

```

0169             'experiment_desc', sinfo.experiment_description, ...
```

```

0170             'reference_ids', refIds, ...
```

```

0171             'additional_comments', sinfo.additional_comments, ...
```

```

0172             'additional_authors', sinfo.additional_authors, ...
```

```

0173             'keywords', sinfo.keywords, ...
```

```

0174             'quantity', sinfo.quantity, ...
```

```

0175             'analysis_desc', sinfo.analysis_description...
0176         );
```

```

0177         utils.helper.msg(msg.PROC1, 'made meta-data entry');
```

```

0178     catch
```

```

0179         lasterr
```

```

0180         error('### failed to make meta-data entry');
```

```

0181     end
```

```

0182     %%%%%%%%%%% Update object tables
```

```

0183     eid = update_object_tables(conn, obj(jj), objid);
```

```

0184     %%%%%%%%%%% Update transactions table
```

```

0185     try
```

```

0186         message = utils.mysql.insert(conn, ...
```

```

0187             'transactions', ...
```

```

0188             'obj_id', objid, ...
```

```

0189             'user_id', userid, ...
```

```

0190             'transdate', tdate, ...
```

```

0191             'direction', 'in'...
0192         );
```

```

0193         utils.helper.msg(msg.PROC1, 'updated transactions table');
```

```

0194     catch
```

```

0195         error('### Failed to update transactions table');
```

```

0196     end
```

```

0197     % Collect this ID
```

```

0198     ids = [ids objid];
```

```

0199     end
```

```

0200     end
```

```

0201     %% Now make collection entry
```

```

0202     if ~isempty(ids)
```

```

0203         try
```

```

0204             message = utils.mysql.insert(conn, ...
```

```

0205                 'collections', ...
```

```

0206                 'nobjs', length(ids), ...
```

```

0207                 'obj_ids', csv(ids)...
```

```

0208             );
```

```

0209             utils.helper.msg(msg.PROC1, 'made collection entry');
```

```

0210         catch
```

```

0211             error('### Failed to make an entry in the collections table');
```

```

0212         end
```

```

0213     end
```

```

0214     catch ME
```

```

0215         utils.helper.msg(msg.PROC1, '### Submission error - cleaning up.');
```

```

0216         rethrow(ME)
```

```

0217     end
```

```

0218     % Pass back output
```

```

0219     varargout{1} = ids;
```

```

0220     varargout{2} = getCollectionID(conn, ids);
```

```

0221     utils.helper.msg(msg.PROC1, 'submission complete.');
```

```

0232 end
0233
0234 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0235 %                                     Local Functions                                     %
0236 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0237
0238 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0239 % Update entries for various object tables
0240 %
0241 % eid = update_object_tables(conn, obj)
0242 % eid = update_object_tables(conn, obj, objid)
0243 %
0244 function eid = update_object_tables(varargin)
0245
0246     import utils.const.*
0247
0248     if nargin < 2
0249         error('### Incorrect usage.');
```



```

0318         message = utils.mysql.insert(conn, ...
0319         'fsdata', ...
0320         'xunits', char(obj.xunits), ...
0321         'yunits', char(obj.yunits));
0322     end
0323     utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
0324     % get the entry id
0325     eid = getObjID(conn, '');
0326 catch
0327     lasterr
0328     error('### failed to make meta-data entry for %s table', cl);
0329 end
0330 %----- MFIR
0331 case 'mfir'
0332 % Make the meta-data entry
0333 try
0334     if isempty(obj.fs)
0335         message = utils.mysql.insert(conn, ...
0336         'mfir', ...
0337         'obj_id', objid, ...
0338         'in_file', obj.infile ...
0339         );
0340     else
0341         message = utils.mysql.insert(conn, ...
0342         'mfir', ...
0343         'obj_id', objid, ...
0344         'in_file', obj.infile, ...
0345         'fs', obj.fs ...
0346         );
0347     end
0348     utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
0349     % get the entry id
0350     eid = getObjID(conn, '');
0351 catch
0352     lasterr
0353     error('### failed to make meta-data entry for %s table', cl);
0354 end
0355 %----- MIIR
0356 case 'miir'
0357 % Make the meta-data entry
0358 try
0359     if isempty(obj.fs)
0360         message = utils.mysql.insert(conn, ...
0361         'miir', ...
0362         'obj_id', objid, ...
0363         'in_file', obj.infile ...
0364         );
0365     else
0366         message = utils.mysql.insert(conn, ...
0367         'miir', ...
0368         'obj_id', objid, ...
0369         'in_file', obj.infile, ...
0370         'fs', obj.fs ...
0371         );
0372     end
0373     utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
0374     % get the entry id
0375     eid = getObjID(conn, '');
0376 catch
0377     lasterr
0378     error('### failed to make meta-data entry for %s table', cl);
0379 end
0380 %----- TSDATA
0381 case 'tsdata'
0382 % Make the meta-data entry
0383 try
0384     message = utils.mysql.insert(conn, ...
0385     'tsdata', ...
0386     'xunits', char(obj.xunits), ...
0387     'yunits', char(obj.yunits), ...
0388     'fs', obj.fs, ...
0389     'nsecs', obj.nsecs, ...
0390     't0', format(obj.t0, 'yyyy-mm-dd HH:MM:SS') ...
0391     );
0392     utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
0393     % get the entry id
0394     eid = getObjID(conn, '');
0395 catch
0396     lasterr
0397     error('### failed to make meta-data entry for %s table', cl);
0398 end
0399 %----- XYDATA
0400 case 'xydata'
0401 % Make the meta-data entry
0402 try
0403     message = utils.mysql.insert(conn, ...

```

```

0404         'xydata', ...
0405         'xunits', char(obj.xunits), ...
0406         'yunits', char(obj.yunits), ...
0407     );
0408     utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
0409     % get the entry id
0410     eid = getObjID(conn, '');
0411     catch
0412         lasterr
0413         error('### failed to make meta-data entry for %s table', cl);
0414     end
0415     otherwise
0416         utils.helper.msg(msg.PROC2, '# no special table for objects of type %s', cl);
0417     end
0418
0419     if ~isempty(eid)
0420         utils.helper.msg(msg.PROC2, 'made meta-data entry in %s table with id %d', cl,
0421         eid);
0422     end
0423
0424     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0425     %
0426     function id = getObjID(conn, hash)
0427
0428         try
0429             q = sprintf('select last_insert_id()');
0430             curs = exec(conn, q);
0431             curs = fetch(curs);
0432             id = curs.Data{1};
0433             close(curs);
0434         catch
0435             error('### Failed to get object ID. Server returned %s', curs.Message);
0436         end
0437     end
0438
0439     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0440     %
0441     function Cid = getCollectionID(conn, objids)
0442
0443         try
0444             curs = exec(conn, sprintf('select id from collections where obj_ids="%s"',
0445             csv(objids)));
0446             curs = fetch(curs);
0447             Cid = curs.Data{1};
0448             close(curs);
0449         catch
0450             error('### Failed to get Collection ID. Server returned %s', curs.Message);
0451         end
0452
0453         if strcmp(Cid, 'No Data')
0454             Cid = [];
0455         end
0456     end
0457
0458     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0459     % make comma separated list of numbers
0460     function s = csv(x)
0461
0462         s = sprintf('%g,', x);
0463         s = s(1:end-1);
0464     end
0465
0466     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0467     % check sinfo structure
0468     %
0469     function sinfo = check_sinfo(sinfo)
0470
0471         import utils.const.*
0472
0473         % --- check fieldnames
0474         manfields = {'conn', 'experiment_title', 'experiment_description',
0475         'analysis_description'};
0476         allfields = {'conn', 'experiment_title', 'experiment_description',
0477         'analysis_description', ...
0478         'quantity', 'keywords', 'reference_ids', 'additional_comments',
0479         'additional_authors'};
0480
0481         if ~isstruct(sinfo)
0482             sinfo_description();
0483         end
0484
0485         % get input fieldnames
0486         fnames = fieldnames(sinfo);
0487     end

```

```

0485 % check these
0486 for jj=1:length(manfields)
0487     if ~ismember(fnames, manfields{jj})
0488         for kk=1:length(manfields)
0489             utils.helper.msg(msg.OFF, 'Required Field: %s', manfields{kk});
0490         end
0491         error('The sinfo structure should contain at least the above fields.');
```

```

0492     end
0493 end
0494
0495 % Check other fields exist
0496 if ~isfield(sinfo, 'quantity')
0497     sinfo.quantity = '';
0498 end
0499 if ~isfield(sinfo, 'keywords')
0500     sinfo.keywords = '';
0501 end
0502 if ~isfield(sinfo, 'reference_ids')
0503     sinfo.reference_ids = '';
0504 end
0505 if ~isfield(sinfo, 'additional_comments')
0506     sinfo.additional_comments = '';
0507 end
0508 if ~isfield(sinfo, 'additional_authors')
0509     sinfo.additional_authors = '';
0510 end
0511
0512 %---- Additional checks
0513 if length(sinfo.experiment_title) < 5
0514     error('### Experiment title is mandatory and should be at least 5 characters
long.');
```

```

0515 end
0516 if length(sinfo.experiment_description) < 10
0517     error('### Experiment description is mandatory and should be more that 10
characters long.');
```

```

0518 end
0519 if length(sinfo.analysis_description) < 10
0520     error('### Analysis description is mandatory and should be more that 10 characters
long.');
```

```

0521 end
0522
0523 % check all fields are present otherwise make them empty
0524 for jj=1:length(allfields)
0525     if ~ismember(fnames, allfields{jj})
0526         utils.helper.msg(msg.PROC2, 'setting default for field %s', allfields{jj});
0527         sinfo.(allfields{jj}) = '';
0528     end
0529 end
0530
0531 utils.helper.msg(msg.PROC1, 'sinfo structure is valid.');
```

```

0532 end
0533
0534
0535 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0536 % Throw description of sinfo to terminal
0537 %
0538 function sinfo_description
0539     disp('The sinfo structure should look like:');
0540     disp(' ');
0541     disp(' sinfo.conn - a database connection object');
0542     disp(' sinfo.experiment_title - the experiment title (Mandatory, >5
characters)');
0543     disp(' sinfo.experiment_description - the experiment description (Mandatory, >10
characters)');
0544     disp(' sinfo.analysis_description - the analysis description (Mandatory, >10
characters)');
0545     disp(' sinfo.quantity - the physical quantity represented by the
data');
```

```

0546     disp(' sinfo.keywords - a comma-delimited list of keywords');
0547     disp(' sinfo.reference_ids - IDs of other applicable LTPDA objects in the
repository');
0548     disp(' sinfo.additional_comments - any additional comments');
0549     disp(' sinfo.additional_authors - any additional authors');
0550     disp(' ');
0551     error('### Incorrect specification for sinfo.')
```

```

0552 end
0553
0554 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0555 %
0556 function ii = getInfo(varargin)
0557     if nargin == 1 && strcmpi(varargin{1}, 'None')
0558         sets = {};
0559         pl = [];
0560     else
0561         sets = {'Default'};
0562         pl = getDefaultPlist;
```

```
0563     end
0564     % Build info object
0565     ii = minfo(mfilename, 'ltpda_uo', '', utils.const.categories.internal, '$Id:
submit.m,v 1.6 2008/09/04 15:29:30 ingo Exp $', sets, pl);
0566 end
0567
0568 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0569 %
0570 function plo = getDefaultPlist()
0571     sinfo.conn = [];
0572     sinfo.experiment_title = 'please add title';
0573     sinfo.experiment_description = 'please add experiment description';
0574     sinfo.analysis_description = 'please add description';
0575     sinfo.quantity = 'none';
0576     sinfo.keywords = 'none';
0577     sinfo.reference_ids = '';
0578     sinfo.additional_comments = 'none';
0579     sinfo.additional_authors = 'no one';
0580     plo = plist('sinfo', sinfo);
0581 end
0582
```

Contents

PURPOSE ^

%% class: ltpda_uoh %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ltpda_uoh %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help ltpda_uoh/addHistory">ltpda_uoh/addHistory</a>      - Add a history-
    object to the ltpda_uo object.
    <a href="matlab:help ltpda_uoh/fromDataInMAT">ltpda_uoh/fromDataInMAT</a> - Default
    method to convert a data-array into am ltpda_uoh
    <a href="matlab:help ltpda_uoh/fromDatafile">ltpda_uoh/fromDatafile</a>   - Default
    method to convert a data-file into a ltpda_uoh-object
    <a href="matlab:help ltpda_uoh/fromFile">ltpda_uoh/fromFile</a>           - Construct a
    ltpda_ob from a file
    <a href="matlab:help ltpda_uoh/fromLISO">ltpda_uoh/fromLISO</a>           - Default method to
    read LISO files
    <a href="matlab:help ltpda_uoh/fromRepository">ltpda_uoh/fromRepository</a> - Retrieve a
    ltpda_uo from a repository
    <a href="matlab:help ltpda_uoh/ltpda_uoh">ltpda_uoh/ltpda_uoh</a>         - is the abstract
    ltpda base class for ltpda user object classes with history
    <a href="matlab:help ltpda_uoh/setHist">ltpda_uoh/setHist</a>             - Set the property
    'hist'
    <a href="matlab:help ltpda_uoh/setName">ltpda_uoh/setName</a>             - Set the property
    'name'.
    <a href="matlab:help ltpda_uoh/type">ltpda_uoh/type</a>                 - converts the input
    objects to MATLAB functions.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

fromDataInMAT

PURPOSE ^

FROMDATAINMAT Default method to convert a data-array into am ltpda_uoh

SYNOPSIS ^

function obj = fromDataInMAT(obj, data, pli)

DESCRIPTION ^

FROMDATAINMAT Default method to convert a data-array into am ltpda_uoh
%%%

FUNCTION: fromDataInMAT

DESCRIPTION: Convert a saved data-array into an AO with a tsdata-object

CALL: obj = fromDataInMAT(obj, data-array, plist)

PARAMETER: data-array: data-array
 plist: plist-object (must contain the filename)

VERSION: \$Id: fromDataInMAT.m,v 1.1 2008/08/22 14:17:26 ingo Exp \$

HISTORY: 22-03-2008 M Hewitson
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SOURCE CODE ^

```
0001 % FROMDATAINMAT Default method to convert a data-array into am ltpda_uoh
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:        fromDataInMAT
0005 %
0006 % DESCRIPTION: Convert a saved data-array into an AO with a tsdata-object
0007 %
0008 % CALL:            obj = fromDataInMAT(obj, data-array, plist)
0009 %
0010 % PARAMETER:      data-array: data-array
0011 %                   plist:        plist-object (must contain the filename)
0012 %
0013 % VERSION:        $Id: fromDataInMAT.m,v 1.1 2008/08/22 14:17:26 ingo Exp $
0014 %
0015 % HISTORY:        22-03-2008 M Hewitson
0016 %                   Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 function obj = fromDataInMAT(obj, data, pli)
0020
0021     error('### This mat-file doesn''t contain a %s-object', class(obj));
0022
0023 end
```


fromDatafile

PURPOSE ^

FROMDATAFILE Default method to convert a data-file into a ltpda_uoh-object

SYNOPSIS ^

function obj = fromDatafile(obj, pli)

DESCRIPTION ^

```
FROMDATAFILE Default method to convert a data-file into a ltpda_uoh-object
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromDatafile

DESCRIPTION:   Default method to convert a data-file into a ltpda_uoh-object

CALL:         obj = fromDataInMAT(obj, pli)

PARAMETER:    obj: empty ltpda_uoh-object
               pli: plist-object (must contain the filename)

VERSION:      $Id: fromDatafile.m,v 1.1 2008/08/22 14:17:26 ingo Exp $

HISTORY:      22-03-2008 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SOURCE CODE ^

```
0001 % FROMDATAFILE Default method to convert a data-file into a ltpda_uoh-object
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromDatafile
0005 %
0006 % DESCRIPTION:   Default method to convert a data-file into a ltpda_uoh-object
0007 %
0008 % CALL:         obj = fromDataInMAT(obj, pli)
0009 %
0010 % PARAMETER:    obj: empty ltpda_uoh-object
0011 %               pli: plist-object (must contain the filename)
0012 %
0013 % VERSION:      $Id: fromDatafile.m,v 1.1 2008/08/22 14:17:26 ingo Exp $
0014 %
0015 % HISTORY:      22-03-2008 M Hewitson
0016 %               Creation
0017 %
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019
0020 function obj = fromDatafile(obj, pli)
0021
0022     error('### It is not possible to convert this data file into a %s-object',
class(obj));
0023
0024 end
```


fromFile

PURPOSE ^

Construct a ltpda_ob from a file

SYNOPSIS ^

function objs = fromFile(obj, pli)

DESCRIPTION ^

```
Construct a ltpda_ob from a file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

FUNCTION:      fromFile

DESCRIPTION: Construct a ltpda_ob from a file

CALL:         obj = obj.fromFile(filename)
              obj = obj.fromFile(pl)

VERSION:      $Id: fromFile.m,v 1.3 2008/09/08 09:40:46 hewitson Exp $

HISTORY:      21-08-2008 Diepholz
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SOURCE CODE ^

```
0001 % Construct a ltpda_ob from a file
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromFile
0005 %
0006 % DESCRIPTION: Construct a ltpda_ob from a file
0007 %
0008 % CALL:         obj = obj.fromFile(filename)
0009 %              obj = obj.fromFile(pl)
0010 %
0011 % VERSION:      $Id: fromFile.m,v 1.3 2008/09/08 09:40:46 hewitson Exp $
0012 %
0013 % HISTORY:      21-08-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 function objs = fromFile(obj, pli)
0018
0019     VERSION = '$Id: fromFile.m,v 1.3 2008/09/08 09:40:46 hewitson Exp $';
0020
0021     % Which file type are we dealing with?
0022     if ischar(pli)
0023         pli = plist('filename', pli);
0024     end
0025
0026     % get filename
0027     filename = find(pli, 'filename');
0028
```

```

0029 % Get the correct parameter set
0030 [path, name, ext] = fileparts(filename);
0031
0032 switch ext
0033
0034     case '.fil'
0035         objs = obj.fromLISO(pli);
0036
0037     case '.mat'
0038         ii = obj.getInfo(class(obj), 'From MAT File');
0039         ii.setMversion([VERSION '-->' ii.mversion]);
0040         objs = load(filename);
0041
0042         if isstruct(objs) && isfield(objs, 'objs')
0043             scl = utils.helper.classFromStruct(objs.objs);
0044             if isempty(scl)
0045                 error('### The structure does not match any LTPDA object.');
```

0046 end

0047 if ~strcmp(class(obj), scl)

0048 error('### The structure does not match the chosen LTPDA object constructor.

It seems to be a [%s] object.', scl)

0049 end

0050 objs = feval(class(obj), objs.objs);

0051 objs.addHistory(ii, pli, [], []);

0052 elseif ismember('a', fieldnames(objs))

0053 objs = objs.a;

0054 objs.addHistory(ii, pli, [], []);

0055 elseif ismember('objs', fieldnames(objs))

0056 objs = objs.objs;

0057 objs.addHistory(ii, pli, [], []);

0058 else

0059 objs = obj.fromDataInMAT(objs, pli);

0060 end

0061

0062 case '.xml'

0063 root_node = xmlread(filename);

0064 objs = utils.helper.xmlread(root_node, class(obj));

0065

0066 case {' .txt', '.dat'}

0067 % Add default values

0068 objs = obj.fromDatafile(pli);

0069

0070 otherwise

0071 error('### Unknown file type [%s].', ext(2:end));

0072 end

0073

0074 end

0075

fromLISO

PURPOSE ^

FROMLISO Default method to read LISO files

SYNOPSIS ^

function obj = fromLISO(obj, pli)

DESCRIPTION ^

```
FROMLISO Default method to read LISO files
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FUNCTION:      fromLISO
DESCRIPTION:   If a class can not handle a LISO file then throw always an error.
               If a class can handle thie file then overload this method.
CALL:         f = fromLISO(f, pli)
PARAMETER:    pzm: Empty ltpda_uoh-object
               pli: input plist (must contain the filename)
VERSION:      $Id: fromLISO.m,v 1.1 2008/08/22 14:17:26 ingo Exp $
HISTORY:      21-08-2008 Diepholz
               Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

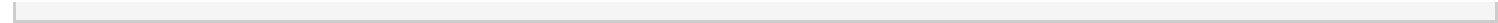
This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SOURCE CODE ^

```
0001 % FROMLISO Default method to read LISO files
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:      fromLISO
0005 %
0006 % DESCRIPTION:   If a class can not handle a LISO file then throw always an error.
0007 %               If a class can handle thie file then overload this method.
0008 %
0009 % CALL:         f = fromLISO(f, pli)
0010 %
0011 % PARAMETER:    pzm: Empty ltpda_uoh-object
0012 %               pli: input plist (must contain the filename)
0013 %
0014 % VERSION:      $Id: fromLISO.m,v 1.1 2008/08/22 14:17:26 ingo Exp $
0015 %
0016 % HISTORY:      21-08-2008 Diepholz
0017 %               Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function obj = fromLISO(obj, pli)
0022
0023     error('### It is not possible to convert a LISO file into a %s-object', class(obj));
0024
0025 end
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

fromRepository

PURPOSE ^

Retrieve a ltpda_uo from a repository

SYNOPSIS ^

function objs = fromRepository(obj, pli)

DESCRIPTION ^

Retrieve a ltpda_uo from a repository

FUNCTION: fromRepository

DESCRIPTION: Retrieve a ltpda_uo from a repository

CALL: obj = fromRepository(pl)

PARAMETER: pl: Parameter list object

VERSION: \$Id: fromRepository.m,v 1.3 2008/09/04 13:37:14 ingo Exp \$

HISTORY: 07-05-2007 Hewitson
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SOURCE CODE ^

```
0001 % Retrieve a ltpda_uo from a repository
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION: fromRepository
0005 %
0006 % DESCRIPTION: Retrieve a ltpda_uo from a repository
0007 %
0008 % CALL: obj = fromRepository(pl)
0009 %
0010 % PARAMETER: pl: Parameter list object
0011 %
0012 % VERSION: $Id: fromRepository.m,v 1.3 2008/09/04 13:37:14 ingo Exp $
0013 %
0014 % HISTORY: 07-05-2007 Hewitson
0015 % Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 function objs = fromRepository(obj, pli)
0019
0020     VERSION = '$Id: fromRepository.m,v 1.3 2008/09/04 13:37:14 ingo Exp $';
0021
0022     % get object info
0023     ii = obj.getInfo(class(obj), 'From Repository');
0024     % Set the method version string in the minfo object
0025     ii.setMversion([VERSION '-->' ii.mversion]);
0026
0027     % Add default values
```

```

0028 pl = combine(pli, ii.plist);
0029
0030 % Get parameters
0031 conn = find(pl, 'conn');
0032 hostname = find(pl, 'hostname');
0033 database = find(pl, 'database');
0034 ids = find(pl, 'id');
0035 cid = find(pl, 'cid');
0036
0037 % do we have a connection?
0038 closeConn = 0;
0039 if isempty(conn)
0040     closeConn = 1;
0041     % Connect to repository
0042     conn = utils.mysql.connect(hostname, database);
0043 end
0044 if ~isa(conn, 'database')
0045     error('### connection to the database failed.');
```

```

0046 end
0047
0048 if isempty(ids) && ~isempty(cid)
0049     % get the ids from the cid
0050     ids = utils.mysql.getObjIds(conn, cid);
0051 end
0052
0053 % Get each ID
0054 Nids = length(ids);
0055 objs = [];
0056
0057 for kk=1:Nids
0058
0059     %---- This id
0060     id = ids(kk);
0061     utils.helper.msg(utils.const.msg.OPROC2, 'retrieving ID %d', id);
0062
0063     %---- check ID object type
0064     tt = utils.mysql.getObjType(conn, id);
0065
0066     %---- If this is an AO
0067     if strcmp(tt, class(obj))
0068         %---- call database constructor
0069         obj = ltpda_uo.retrieve(conn, id);
0070         %---- Add history
0071
0072         % make a new history object
0073         % - remove connection object from plist first
0074         txt = textscan(conn.URL, '%s', 'Delimiter', '/', 'CollectOutput', 1);
0075         plc = plist('hostname', txt{1}{3}, 'database', txt{1}{4});
0076         pl = combine(pli, plc);
0077         pl.remove('conn');
0078         obj.addHistory(ii, pl, [], obj.hist);
0079
0080         %---- Add to output array
0081         objs = [objs obj];
0082     else
0083         warning('      !skipping ID %d, type %s', id, tt);
0084     end
0085 end
0086
0087
0088 % close connection
0089 if closeConn
0090     close(conn);
0091 end
0092 end
0093

```


setHist

PURPOSE ^

SETHIST Set the property 'hist'

SYNOPSIS ^

function obj = setHist(obj, val)

DESCRIPTION ^

SETHIST Set the property 'hist'

DESCRIPTION: SETHIST Set the property 'hist'

CALL: obj = obj.setHist('new ver');
obj = setHist(obj, 'new ver');

INPUTS: obj - is a general object

VERSION: \$Id: setHist.m,v 1.2 2008/07/18 11:13:34 ingo Exp \$

HISTORY: 16-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_uoh](#) LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history

SOURCE CODE ^

```
0001 % SETHIST Set the property 'hist'
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: SETHIST Set the property 'hist'
0005 %
0006 % CALL:      obj = obj.setHist('new ver');
0007 %           obj = setHist(obj, 'new ver');
0008 %
0009 % INPUTS:    obj - is a general object
0010 %
0011 % VERSION:   $Id: setHist.m,v 1.2 2008/07/18 11:13:34 ingo Exp $
0012 %
0013 % HISTORY:   16-07-2008 Diepholz
0014 %           Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017
0018 function obj = setHist(obj, val)
0019     %% decide whether we modify the ltpda_uoh-object, or create a new one.
0020     obj = copy(obj, nargout);
0021
0022     %% set 'hist'
0023     obj.hist = val;
0024 end
```


Contents

PURPOSE ^

%% class: mfir %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: mfir %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help mfir/char">mfir/char</a>                - convert a mfir object into a
    string.
    <a href="matlab:help mfir/copy">mfir/copy</a>                - Make copy of mfir objects
    depending of the second input
    <a href="matlab:help mfir/display">mfir/display</a>          - overloads display
    functionality for mfir objects.
    <a href="matlab:help mfir/fromA">mfir/fromA</a>              - Construct an mfir from
    coefficients
    <a href="matlab:help mfir/fromAO">mfir/fromAO</a>            - create FIR filter from
    magnitude of input AO/fsdata
    <a href="matlab:help mfir/fromPzmodel">mfir/fromPzmodel</a>  - Construct an mfir from
    a pzmodel
    <a href="matlab:help mfir/fromStandard">mfir/fromStandard</a> - Construct an mfir
    from a standard types
    <a href="matlab:help mfir/mfir">mfir/mfir</a>                - FIR filter object class
    constructor.
    <a href="matlab:help mfir/mkbandpass">mfir/mkbandpass</a>    - return a bandpass
    filter mfir(). A Cheby filter is used.
    <a href="matlab:help mfir/mkbandreject">mfir/mkbandreject</a> - return a low pass
    filter mfir(). A Butterworth filter is used.
    <a href="matlab:help mfir/mkhighpass">mfir/mkhighpass</a>    - return a high pass
    filter mfir(). A Butterworth filter is used.
    <a href="matlab:help mfir/mklowpass">mfir/mklowpass</a>      - return a low pass
    filter mfir().
    <a href="matlab:help mfir/parseFilterParams">mfir/parseFilterParams</a> - parses the
    input plist and returns a full plist for designing a standard FIR filter.
    <a href="matlab:help mfir/redesign">mfir/redesign</a>        - redesign the input
    filter to work for the given sample rate.
    <a href="matlab:help mfir/resp">mfir/resp</a>                - Make a frequency response of
    the filter.
    <a href="matlab:help mfir/setGd">mfir/setGd</a>              - Set the property 'gd'
    <a href="matlab:help mfir/string">mfir/string</a>            - writes a command string
    that can be used to recreate the input filter object.
    <a href="matlab:help mfir/update_struct">mfir/update_struct</a> - update the input
    structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: miir %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: miir %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help miir/char">miir/char</a>                - convert a miir object into a
    string.
    <a href="matlab:help miir/copy">miir/copy</a>                - Make copy of miir objects
    depending of the second input
    <a href="matlab:help miir/display">miir/display</a>          - overloads display
    functionality for miir objects.
    <a href="matlab:help miir/filload">miir/filload</a>          - % Load a LISO *_iir.fil
    file to get the filter taps and return a
    <a href="matlab:help miir/fromAB">miir/fromAB</a>            - Construct an miir from
    coefficients
    <a href="matlab:help miir/fromLISO">miir/fromLISO</a>        - Construct a miir filter
    from a LISO file
    <a href="matlab:help miir/fromPzmodel">miir/fromPzmodel</a>  - Construct an miir from
    a pzmodel
    <a href="matlab:help miir/fromStandard">miir/fromStandard</a> - Construct an miir
    from a standard types
    <a href="matlab:help miir/miir">miir/miir</a>                - IIR filter object class
    constructor.
    <a href="matlab:help miir/mkbandpass">miir/mkbandpass</a>    - return a bandpass
    filter miir(). A Cheby filter is used.
    <a href="matlab:help miir/mkbandreject">miir/mkbandreject</a> - return a low pass
    filter miir(). A Butterworth filter is used.
    <a href="matlab:help miir/mkhighpass">miir/mkhighpass</a>    - return a high pass
    filter miir(). A Butterworth filter is used.
    <a href="matlab:help miir/mklowpass">miir/mklowpass</a>      - return a low pass
    filter miir(). A Butterworth filter is used.
    <a href="matlab:help miir/parseFilterParams">miir/parseFilterParams</a> - parses the
    input plist and returns a full plist for designing a standard IIR filter.
    <a href="matlab:help miir/redesign">miir/redesign</a>        - redesign the input
    filter to work for the given sample rate.
    <a href="matlab:help miir/resp">miir/resp</a>                - Make a frequency response of
    the filter.
    <a href="matlab:help miir/setB">miir/setB</a>                - Set the property 'b'
    <a href="matlab:help miir/setHistin">miir/setHistin</a>        - Set the property
    'histin'
    <a href="matlab:help miir/string">miir/string</a>            - writes a command string
    that can be used to recreate the input filter object.
    <a href="matlab:help miir/update_struct">miir/update_struct</a> - update the input
    structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: minfo %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: minfo %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help minfo/char">minfo/char</a>           - convert an minfo object into a
string.
    <a href="matlab:help minfo/copy">minfo/copy</a>           - Make copy of minfo objects
depending of the second input
    <a href="matlab:help minfo/display">minfo/display</a>     - display an minfo object.
    <a href="matlab:help minfo/minfo">minfo/minfo</a>         - a helper class for LTPDA
methods.
    <a href="matlab:help minfo/setMversion">minfo/setMversion</a> - Set the property
'mversion'.
    <a href="matlab:help minfo/update_struct">minfo/update_struct</a> - update the input
structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: param %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: param %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help param/char">param/char</a>          - convert a param object into a
string.
    <a href="matlab:help param/copy">param/copy</a>          - Make copy of param objects
depending of the second input
    <a href="matlab:help param/display">param/display</a>      - display a parameter
    <a href="matlab:help param/display2">param/display2</a>    - display a parameter
    <a href="matlab:help param/mux">param/mux</a>             - concatenate params into a
vector.
    <a href="matlab:help param/param">param/param</a>         - Parameter object class
constructor.
    <a href="matlab:help param/setKey">param/setKey</a>        - Set the property 'key'.
    <a href="matlab:help param/setKeyVal">param/setKeyVal</a>  - Set the properties 'key'
and 'val'
    <a href="matlab:help param/setVal">param/setVal</a>        - Set the property 'val'.
    <a href="matlab:help param/string">param/string</a>        - writes a command string that
can be used to recreate the input param object.
    <a href="matlab:help param/update_struct">param/update_struct</a> - update the input
structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: plist %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%% class: plist %%%

plist/append
plist-object or a key/value pair to the parameter list.
plist/char
a string.
plist/combine
(plist objects) into a single plist.
plist/copy
depending of the second input
plist/display
plist/find
parameter list.
plist/fromFile
a file
plist/fromRepository
filter from a repository
plist/isparam
the parameter lists.
plist/nparams
param objects in the list.
plist/plist
constructor.
plist/pset
a param-object into the parameter list.
plist/remove
parameter list.
plist/resp
pzmodel/resp.
plist/string
a command string which will recreate the plist object.
plist/update_struct

- append a param-object,
- convert a parameter list into
- multiple parameter lists
- Make copy of plist objects
- display plist object.
- overloads find routine for a
- Construct a ltpda_ob from
- Construct a plist
- look for a given key in
- returns the number of
- Plist class object
- set or add a key/value pairor
- remove a parameter from the
- shadows miir/iirResp and
- converts a plist object to
- update the input

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: provenance %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: provenance %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help provenance/char">provenance/char</a>           - convert a provenance
    object into a string.
    <a href="matlab:help provenance/copy">provenance/copy</a>           - Make copy of
    provenance objects depending of the second input
    <a href="matlab:help provenance/display">provenance/display</a>       - overload terminal
    display for provenance objects.
    <a href="matlab:help provenance/provenance">provenance/provenance</a> - constructors
    for provenance class.
    <a href="matlab:help provenance/string">provenance/string</a>         - writes a command
    string that can be used to recreate the input provenance object.
    <a href="matlab:help provenance/update_struct">provenance/update_struct</a> - update the
    input structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: pz %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: pz %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help pz/char">pz/char</a>           - convert a pz object into a string.
    <a href="matlab:help pz/copy">pz/copy</a>           - Make copy of pz objects depending of
the second input
    <a href="matlab:help pz/cp2iir">pz/cp2iir</a>       - Return a,b IIR filter coefficients
for a complex pole designed using the bilinear transform.
    <a href="matlab:help pz/cz2iir">pz/cz2iir</a>       - return a,b IIR filter coefficients
for a complex zero designed using the bilinear transform.
    <a href="matlab:help pz/display">pz/display</a>     - display a pz object.
    <a href="matlab:help pz/fq2ri">pz/fq2ri</a>         - Convert frequency/Q pole/zero
representation
    <a href="matlab:help pz/pz">pz/pz</a>               - is the ltpda class that provides a
common definition of poles and zeros.
    <a href="matlab:help pz/resp">pz/resp</a>           - returns the complex response of the
pz object.
    <a href="matlab:help pz/ri2fq">pz/ri2fq</a>         - Convert complx pole/zero into
frequency/Q pole/zero representation.
    <a href="matlab:help pz/rp2iir">pz/rp2iir</a>       - Return a,b coefficients for a real
pole designed using the bilinear transform.
    <a href="matlab:help pz/rz2iir">pz/rz2iir</a>       - Return a,b IIR filter coefficients
for a real zero designed using the bilinear transform.
    <a href="matlab:help pz/setF">pz/setF</a>           - Set the property 'f'
    <a href="matlab:help pz/setQ">pz/setQ</a>           - Set the property 'q'
    <a href="matlab:help pz/setRI">pz/setRI</a>         - Set the property 'ri' and computes
'f' and 'q'
    <a href="matlab:help pz/string">pz/string</a>       - writes a command string that can
be used to recreate the input pz object.
    <a href="matlab:help pz/update_struct">pz/update_struct</a> - update the input structure
to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: pzmodel %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: pzmodel %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help pzmodel/abccascade">pzmodel/abccascade</a>           - Cascade two
filters together to get a new filter.
    <a href="matlab:help pzmodel/char">pzmodel/char</a>                       - convert a pzmodel
object into a string.
    <a href="matlab:help pzmodel/copy">pzmodel/copy</a>                       - Make copy of pzmodel
objects depending of the second input
    <a href="matlab:help pzmodel/display">pzmodel/display</a>                 - overloads display
functionality for pzmodel objects.
    <a href="matlab:help pzmodel/fngen">pzmodel/fngen</a>                     - creates an
arbitrarily long time-series based on the input pzmodel.
    <a href="matlab:help pzmodel/fromLISO">pzmodel/fromLISO</a>               - Construct a
pzmodel from a LISO file
    <a href="matlab:help pzmodel/fromPolesAndZeros">pzmodel/fromPolesAndZeros</a> -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help pzmodel/getlowerFreq">pzmodel/getlowerFreq</a>       - gets the
frequency of the lowest pole or zero in the model.
    <a href="matlab:help pzmodel/getupperFreq">pzmodel/getupperFreq</a>       - gets the
frequency of the highest pole or zero in the model.
    <a href="matlab:help pzmodel/pzm2ab">pzmodel/pzm2ab</a>                   - convert pzmodel to
IIR filter coefficients using bilinear transform.
    <a href="matlab:help pzmodel/pzmodel">pzmodel/pzmodel</a>                 - constructor for
pzmodel class.
    <a href="matlab:help pzmodel/resp">pzmodel/resp</a>                       - returns the complex
response of a pzmodel as an Analysis Object.
    <a href="matlab:help pzmodel/setGain">pzmodel/setGain</a>                 - Set the property
'gain'
    <a href="matlab:help pzmodel/setPoles">pzmodel/setPoles</a>               - Set the property
'poles'
    <a href="matlab:help pzmodel/setZeros">pzmodel/setZeros</a>               - Set the property
'zeros'
    <a href="matlab:help pzmodel/string">pzmodel/string</a>                   - writes a command
string that can be used to recreate the input pzmodel object.
    <a href="matlab:help pzmodel/tomfir">pzmodel/tomfir</a>                   - approximates a
pole/zero model with an FIR filter.
    <a href="matlab:help pzmodel/tomiir">pzmodel/tomiir</a>                   - converts a pzmodel
to an IIR filter using a bilinear transform.
    <a href="matlab:help pzmodel/update_struct">pzmodel/update_struct</a>     - update the
input structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: specwin %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: specwin %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help specwin/char">specwin/char</a>           - convert a specwin object
into a string.
    <a href="matlab:help specwin/copy">specwin/copy</a>           - Make copy of specwin
objects depending of the second input
    <a href="matlab:help specwin/display">specwin/display</a>     - overloads display
functionality for specwin objects.
    <a href="matlab:help specwin/get_window">specwin/get_window</a> - returns the
required window function as a structure.
    <a href="matlab:help specwin/kaiser_alpha">specwin/kaiser_alpha</a> - returns the
alpha parameter that gives the required input
    <a href="matlab:help specwin/kaiser_flatness">specwin/kaiser_flatness</a> - returns the
flatness in dB of the central bin of a kaiser
    <a href="matlab:help specwin/kaiser_nenbw">specwin/kaiser_nenbw</a> - returns the
normalized noise-equivalent bandwidth for a
    <a href="matlab:help specwin/kaiser_rov">specwin/kaiser_rov</a> - returns the
recommended overlap for a Kaiser window with
    <a href="matlab:help specwin/kaiser_w3db">specwin/kaiser_w3db</a> - returns the 3dB
bandwidth in bins of a kaiser window with
    <a href="matlab:help specwin/plot">specwin/plot</a>           - plots a specwin object.
    <a href="matlab:help specwin/specwin">specwin/specwin</a>     - % SPECWIN spectral
window object class constructor.
    <a href="matlab:help specwin/string">specwin/string</a>       - writes a command
string that can be used to recreate the input window object.
    <a href="matlab:help specwin/update_struct">specwin/update_struct</a> - update the
input structure to the current ltpda version
    <a href="matlab:help specwin/win_BH92">specwin/win_BH92</a>    - return BH92 window
sample for input z.
    <a href="matlab:help specwin/win_Bartlett">specwin/win_Bartlett</a> - return Bartlett
window sample for input z.
    <a href="matlab:help specwin/win_FTHP">specwin/win_FTHP</a>    - return FTHP window
sample for input z.
    <a href="matlab:help specwin/win_FTNI">specwin/win_FTNI</a>    - return FTNI window
sample for input z.
    <a href="matlab:help specwin/win_FTSRS">specwin/win_FTSRS</a>  - return FTSRS window
sample for input z.
    <a href="matlab:help specwin/win_HFT116D">specwin/win_HFT116D</a> - return HFT116D
window sample for input z.
    <a href="matlab:help specwin/win_HFT144D">specwin/win_HFT144D</a> - return HFT144D
window sample for input z.
    <a href="matlab:help specwin/win_HFT169D">specwin/win_HFT169D</a> - return HFT169D
window sample for input z.
    <a href="matlab:help specwin/win_HFT196D">specwin/win_HFT196D</a> - return HFT196D
window sample for input z.
    <a href="matlab:help specwin/win_HFT223D">specwin/win_HFT223D</a> - return HFT223D
window sample for input z.
    <a href="matlab:help specwin/win_HFT248D">specwin/win_HFT248D</a> - return HFT248D
window sample for input z.
    <a href="matlab:help specwin/win_HFT70">specwin/win_HFT70</a>  - return HFT70 window
sample for input z.
    <a href="matlab:help specwin/win_HFT90D">specwin/win_HFT90D</a> - return HFT90D
window sample for input z.
    <a href="matlab:help specwin/win_HFT95">specwin/win_HFT95</a>  - return HFT95 window
sample for input z.
    <a href="matlab:help specwin/win_Hamming">specwin/win_Hamming</a> - return Hamming
window sample for input z.
    <a href="matlab:help specwin/win_Hanning">specwin/win_Hanning</a> - return Hanning
window sample for input z.
    <a href="matlab:help specwin/win_Nuttall3">specwin/win_Nuttall3</a> - return Nuttall3
window sample for input z.
    <a href="matlab:help specwin/win_Nuttall3a">specwin/win_Nuttall3a</a> - return
```

```
Nuttall3a window sample for input z.
  <a href="matlab:help specwin/win_Nuttall3b">specwin/win_Nuttall3b</a>    - return
Nuttall3b window sample for input z.
  <a href="matlab:help specwin/win_Nuttall4">specwin/win_Nuttall4</a>      - return Nuttall4
window sample for input z.
  <a href="matlab:help specwin/win_Nuttall4a">specwin/win_Nuttall4a</a>    - return
Nuttall4a window sample for input z.
  <a href="matlab:help specwin/win_Nuttall4b">specwin/win_Nuttall4b</a>    - return
Nuttall4b window sample for input z.
  <a href="matlab:help specwin/win_Nuttall4c">specwin/win_Nuttall4c</a>    - return
Nuttall4c window sample for input z.
  <a href="matlab:help specwin/win_Rectangular">specwin/win_Rectangular</a> - return
rectangular window sample for input z.
  <a href="matlab:help specwin/win_SFT3F">specwin/win_SFT3F</a>            - return SFT3F window
sample for input z.
  <a href="matlab:help specwin/win_SFT3M">specwin/win_SFT3M</a>            - return SFT3M window
sample for input z.
  <a href="matlab:help specwin/win_SFT4F">specwin/win_SFT4F</a>            - return SFT4F window
sample for input z.
  <a href="matlab:help specwin/win_SFT4M">specwin/win_SFT4M</a>            - return SFT4M window
sample for input z.
  <a href="matlab:help specwin/win_SFT5F">specwin/win_SFT5F</a>            - return SFT5F window
sample for input z.
  <a href="matlab:help specwin/win_SFT5M">specwin/win_SFT5M</a>            - return SFT5M window
sample for input z.
  <a href="matlab:help specwin/win_Welch">specwin/win_Welch</a>            - return Welch window
sample for input z.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: ssm %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: ssm %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help ssm/assemble">ssm/assemble</a>           - assembles embedded
subsystems, with exogenous inputs
    <a href="matlab:help ssm/cell_add">ssm/cell_add</a>             - adds corresponding
matrices of same sizes or empty inside cell array
    <a href="matlab:help ssm/cell_fusion">ssm/cell_fusion</a>       - fusions a block
defined matrix stored inside cell array into one matrix
    <a href="matlab:help ssm/cell_mult">ssm/cell_mult</a>          - multiplies block
defined matrix stored inside cell array
    <a href="matlab:help ssm/cell_recut">ssm/cell_recut</a>         - cuts a matrix into
blocks stored inside cell array
    <a href="matlab:help ssm/cell_select">ssm/cell_select</a>       - selects lines and
columns of a block defined matrix stored in
    <a href="matlab:help ssm/cellstrfind">ssm/cellstrfind</a>      - finds matching
strings inside a cellstr array
    <a href="matlab:help ssm/char">ssm/char</a>                   - convert a ssm object into a
string.
    <a href="matlab:help ssm/copy">ssm/copy</a>                     - Make copy of ssm objects
depending of the second input
    <a href="matlab:help ssm/display">ssm/display</a>              - display ssm object.
    <a href="matlab:help ssm/double">ssm/double</a>                - Convert a statespace model
object to double arrays for given i/o
    <a href="matlab:help ssm/isstable">ssm/isstable</a>            - tells if ssm is
numerically stable
    <a href="matlab:help ssm/kalman">ssm/kalman</a>                - applies Kalman filtering
to a discrete ssm with given i/o
    <a href="matlab:help ssm/minreal">ssm/minreal</a>              - gives a minimal
realization of a ssm object by deleting state
    <a href="matlab:help ssm/modifparams">ssm/modifparams</a>      - enables to
options(i_options)y and substitute parameters
    <a href="matlab:help ssm/modiftimestep">ssm/modiftimestep</a>  - modifies the
timestep of a ssm object
    <a href="matlab:help ssm/modify">ssm/modify</a>                - allows to exectue a
string to modify a ssm object
    <a href="matlab:help ssm/reduce">ssm/reduce</a>                - enables to do model
simplification
    <a href="matlab:help ssm/reduce_model">ssm/reduce_model</a>    - enables to do model
simplification
    <a href="matlab:help ssm/simulate">ssm/simulate</a>            - simulates a discrete
ssm with given inputs
    <a href="matlab:help ssm/ssm">ssm/ssm</a>                      - statespace model class
constructor.
    <a href="matlab:help ssm/ssm2iirpz">ssm/ssm2iirpz</a>          - converts a statespace
model object to an miir or a pzmodel
    <a href="matlab:help ssm/ssm2ss">ssm/ssm2ss</a>                - converts a statespace
model object to a MATLAB statespace object.
    <a href="matlab:help ssm/ssmFromBuiltinSystem">ssm/ssmFromBuiltinSystem</a> -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ssm/ssmFromDescription">ssm/ssmFromDescription</a> -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ssm/ssmFromFilename">ssm/ssmFromFilename</a>   -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ssm/ssmFromMiir">ssm/ssmFromMiir</a>         -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ssm/ssmFromPzmodel">ssm/ssmFromPzmodel</a>   -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ssm/ssmFromRepository">ssm/ssmFromRepository</a> -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ssm/ssmFromss">ssm/ssmFromss</a>            -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help ssm/string">ssm/string</a>                 - writes a command string
```

```
that can be used to recreate the input statespace model object.  
  <a href="matlab:help ssm/update_struct">ssm/update_struct</a>      - update the input  
structure to the current ltpda version  
  <a href="matlab:help ssm/validate">ssm/validate</a>      -  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: time %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: time %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help time/char">time/char</a>                - convert a time object into a
    string.
    <a href="matlab:help time/copy">time/copy</a>                - Make copy of time objects
    depending of the second input
    <a href="matlab:help time/display">time/display</a>          - overloads display
    functionality for time objects.
    <a href="matlab:help time/epoch2str">time/epoch2str</a>        - % EPOCH2STR Returns
    computes the time sting of the epoch time stored in 'utc_epoch_milli' and 'created'.
    <a href="matlab:help time/format">time/format</a>            - Returns the time in
    specified format.
    <a href="matlab:help time/fromTimeString">time/fromTimeString</a> - creates a time
    object from a time string.
    <a href="matlab:help time/fromTimeValue">time/fromTimeValue</a> - creates a time
    object from unix epoch time.
    <a href="matlab:help time/getTimezones">time/getTimezones</a> - Get all possible
    timezones.
    <a href="matlab:help time/minus">time/minus</a>              - overloads - operator for
    time objects.
    <a href="matlab:help time/parse_time_string">time/parse_time_string</a> -
    PRASE_TIME_STRING parses the time string.
    <a href="matlab:help time/plus">time/plus</a>                - overloads + operator for
    time objects.
    <a href="matlab:help time/setEpochtime">time/setEpochtime</a> - Set the property
    'utc_epoch_milli'.
    <a href="matlab:help time/setTime_str">time/setTime_str</a>   - Set the property
    'time_str'.
    <a href="matlab:help time/setTimeformat">time/setTimeformat</a> - Set the property
    'timeformat'.
    <a href="matlab:help time/setTimezone">time/setTimezone</a>   - Set the property
    'timezone'.
    <a href="matlab:help time/string">time/string</a>            - writes a command string
    that can be used to recreate the input time object.
    <a href="matlab:help time/time">time/time</a>                - time object class
    constructor.
    <a href="matlab:help time/update_struct">time/update_struct</a> - update the input
    structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: timespan %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: timespan %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help timespan/char">timespan/char</a>                - convert a timespan
    object into a string.
    <a href="matlab:help timespan/computeInterval">timespan/computeInterval</a> - compute the
    interval of the time span.
    <a href="matlab:help timespan/copy">timespan/copy</a>                  - Make copy of time
    objects depending of the second input
    <a href="matlab:help timespan/display">timespan/display</a>            - overloads display
    functionality for timespan objects.
    <a href="matlab:help timespan/fromTimespanDef">timespan/fromTimespanDef</a> -
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help timespan/setEndT">timespan/setEndT</a>          - Set the property
    'endT'.
    <a href="matlab:help timespan/setStartT">timespan/setStartT</a>        - Set the property
    'startT'.
    <a href="matlab:help timespan/setTimeformat">timespan/setTimeformat</a> - Set the
    property 'timeformat'.
    <a href="matlab:help timespan/setTimezone">timespan/setTimezone</a>    - Set the
    property 'timezone'.
    <a href="matlab:help timespan/string">timespan/string</a>              - writes a command
    string that can be used to recreate the input timespan object.
    <a href="matlab:help timespan/timespan">timespan/timespan</a>          - timespan object
    class constructor.
    <a href="matlab:help timespan/update_struct">timespan/update_struct</a> - update the
    input structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: tsdata %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: tsdata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help tsdata/collapseX">tsdata/collapseX</a>      - Checks whether the x
vector is evenly sampled and then removes it
    <a href="matlab:help tsdata/copy">tsdata/copy</a>                  - Make copy of tsdata objects
depending of the second input
    <a href="matlab:help tsdata/display">tsdata/display</a>            - overloads display
functionality for tsdata objects.
    <a href="matlab:help tsdata/fitfs">tsdata/fitfs</a>                - computes the sample rate of
the input data. If the min and max sample
    <a href="matlab:help tsdata/fixNsecs">tsdata/fixNsecs</a>          - fixes the numer of
seconds.
    <a href="matlab:help tsdata/getX">tsdata/getX</a>                  - Get the property 'x'.
    <a href="matlab:help tsdata/growT">tsdata/growT</a>                - grows the time (x) vector
if it is empty.
    <a href="matlab:help tsdata/setFs">tsdata/setFs</a>                - Set the property 'fs'.
    <a href="matlab:help tsdata/setNsecs">tsdata/setNsecs</a>          - Set the property
'nsecs'.
    <a href="matlab:help tsdata/setT0">tsdata/setT0</a>                - Set the property 't0'.
    <a href="matlab:help tsdata/tsdata">tsdata/tsdata</a>              - time-series object class
constructor.
    <a href="matlab:help tsdata/update_struct">tsdata/update_struct</a> - update the input
structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: unit %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: unit %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help unit/char">unit/char</a>          - nameString =
    sprintf('%s[%s^(%g)]',nameString,v.strs{nd},v.exps(nd));
    <a href="matlab:help unit/copy">unit/copy</a>          - Make copy of unit objects
    depending of the second input
    <a href="matlab:help unit/display">unit/display</a>      - display an unit object.
    unit/mpower      - (No help available)
    unit/mrdivide    - (No help available)
    unit/mtimes      - (No help available)
    unit/plus        - (No help available)
    unit/power       - (No help available)
    unit/rdivide     - (No help available)
    unit/sqrt        - (No help available)
    <a href="matlab:help unit/string">unit/string</a>      - converts a unit object to a
    command string which will recreate the unit object.
    unit/times       - (No help available)
    <a href="matlab:help unit/unit">unit/unit</a>          - MINFO a helper class for
    implementing units in LTPDA.
    <a href="matlab:help unit/update_struct">unit/update_struct</a> - update the input
    structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

char

PURPOSE ^

SYNOPSIS ^

```
function str = char(v)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:



plus



string STRING converts a unit object to a command string which will recreate the unit object.

SOURCE CODE ^

```
0001 function str = char(v)
0002     NDimensions = numel(v.strs);
0003     if NDimensions == 0
0004         nameString = '[1]';
0005     else
0006         nameString = '';
0007         for nd = 1:NDimensions
0008             if (v.exps(nd)~=0)
0009                 [n,d] = rat(v.exps(nd));
0010                 if (d==1)
0011                     if v.exps(nd) ~=1
0012                         nameString = sprintf('%s[%s^(%g)]',nameString,v.strs{nd},v.exps(nd));
0013                     else
0014                         nameString = sprintf('%s[%s]',nameString,v.strs{nd});
0015                     end
0016                 else
0017                     nameString = sprintf('%s[%s^(%g/%g)]',nameString,v.strs{nd},n,d);
0018                 end
0019             end
0020         end
0021     end
0022     str = nameString;
0023
0024 end
```


copy

PURPOSE ^

COPY Make copy of unit objects depending of the second input

SYNOPSIS ^

function varargout = copy(old, deepcopy)

DESCRIPTION ^

COPY Make copy of unit objects depending of the second input
DESCRIPTION: COPY Make copy of unit objects depending of the second input
CALL: objs = copy(objs, [1,0]);
M-FILE INFO: Get information about this methods by calling
>> unit.getInfo('copy')
VERSION: \$Id: copy.m,v 1.1 2008/09/05 14:10:15 hewitson Exp \$
HISTORY: 11-07-2008 Diepholz
Creation

CROSS-REFERENCE INFORMATION ^

This function calls:

 unit MINFO a helper class for implementing units in LTPDA.

This function is called by:

-  mpower
-  power
-  sqrt

SOURCE CODE ^

```
0001 % COPY Make copy of unit objects depending of the second input
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: COPY Make copy of unit objects depending of the second input
0005 %
0006 % CALL:      objs = copy(objs, [1,0]);
0007 %
0008 % M-FILE INFO: Get information about this methods by calling
0009 %              >> unit.getInfo('copy')
0010 %
0011 % VERSION:    $Id: copy.m,v 1.1 2008/09/05 14:10:15 hewitson Exp $
0012 %
0013 % HISTORY:    11-07-2008 Diepholz
0014 %              Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function varargout = copy(old, deepcopy)
0019
0020     if deepcopy
0021         ps = properties('unit');
0022         s = size(old);
```

```
0023     obj(s(1),s(2)) = unit;  
0024     for kk=1:numel(old)  
0025         utils.helper.msg(utils.const.msg.OPROC2, 'copying unit');  
0026         for jj=1:length(ps)  
0027             obj(kk).(ps{jj}) = old(kk).(ps{jj});  
0028         end  
0029         % reset on copy  
0030         obj(kk).version = unit.VEROUT;  
0031     end  
0032 else  
0033     obj = old;  
0034 end  
0035 varargout{1} = obj;  
0036 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

display

PURPOSE ^

DISPLAY display an unit object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
DISPLAY display an unit object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: DISPLAY display an unit object.

CALL FOR PARAMETERS:

        obj.display('INFO') % Retrieve method information

VERSION:      $Id: display.m,v 1.1 2008/09/05 14:10:15 hewitson Exp $

HISTORY:      30-01-2007 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function ii = getInfo(varargin)
-  function plo = getDefaultPlist()

SOURCE CODE ^

```
0001 % DISPLAY display an unit object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: DISPLAY display an unit object.
0005 %
0006 % CALL FOR PARAMETERS:
0007 %
0008 %         obj.display('INFO') % Retrieve method information
0009 %
0010 % VERSION:      $Id: display.m,v 1.1 2008/09/05 14:10:15 hewitson Exp $
0011 %
0012 % HISTORY:      30-01-2007 M Hewitson
0013 %               Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 %
0017 function varargout = display(varargin)
0018 %
0019 %%% Check if this is a call for parameters
0020 if utils.helper.isinfocall(varargin{:})
0021     varargout{1} = getInfo(varargin{3});
0022     return
0023 end
0024
```

```
0025 % Get unit objects
0026 objs = utils.helper.collect_objects(varargin(:), 'unit');
0027
0028 % get display text
0029 txt = utils.helper.objdisp(objs);
0030
0031 % display the objects
0032 if nargin > 0
0033     varargout{1} = txt;
0034 elseif nargin == 0;
0035     for j=1:numel(txt)
0036         disp(txt{j});
0037     end
0038 end
0039 end
0040
0041 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0042 %                               Local Functions                               %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0044
0045 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0046 %
0047 % FUNCTION:      getInfo
0048 %
0049 % DESCRIPTION:   Get Info Object
0050 %
0051 % HISTORY:       11-07-07 M Hewitson
0052 %                Creation.
0053 %
0054 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0055
0056 function ii = getInfo(varargin)
0057     if nargin == 1 && strcmpi(varargin{1}, 'None')
0058         sets = {};
0059         pl = [];
0060     else
0061         sets = {'Default'};
0062         pl = getDefaultPlist;
0063     end
0064     % Build info object
0065     ii = minfo(mfilename, 'unit', '', utils.const.categories.output, '$Id: display.m,v 1.1
2008/09/05 14:10:15 hewitson Exp $', sets, pl);
0066 end
0067
0068 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0069 %
0070 % FUNCTION:      getDefaultPlist
0071 %
0072 % DESCRIPTION:   Get Default Plist
0073 %
0074 % HISTORY:       11-07-07 M Hewitson
0075 %                Creation.
0076 %
0077 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0078
0079 function plo = getDefaultPlist()
0080     plo = plist();
0081 end
0082
```

mpower

PURPOSE ^

SYNOPSIS ^

```
function v = mpower(vi, exp)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of unit objects depending of the second input

This function is called by:

SOURCE CODE ^

```
0001
0002 function v = mpower(vi, exp)
0003     v = copy(vi, 1);
0004     if isa(exp, 'unit')
0005         exp = exp.exps;
0006     end
0007     v.exps = v.exps*exp;
0008     v.vals = v.vals^exp;
0009 end
```


mrdivide

PURPOSE ^

SYNOPSIS ^

function v = mrdivide(v1,v2)

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

 [unit](#) MINFO a helper class for implementing units in LTPDA.

This function is called by:

SOURCE CODE ^

```
0001
0002 function v = mrdivide(v1,v2)
0003     v = unit;
0004     v.strs = [v1.strs v2.strs];
0005     v.exps = [v1.exps -1*v2.exps];
0006     v.vals = [v1.vals v2.vals];
0007
0008     v = unit.simplify(v);
0009
0010 end
```


[Home](#) > [classes](#) > [@unit](#) > mtimes.m

mtimes

PURPOSE [^]

SYNOPSIS [^]

```
function v = mtimes(v1,v2)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [unit](#) MINFO a helper class for implementing units in LTPDA.

This function is called by:

SOURCE CODE [^]

```
0001
0002 function v = mtimes(v1,v2)
0003     v = unit;
0004     v.strs = [v1.strs v2.strs];
0005     v.exps = [v1.exps v2.exps];
0006     v.vals = [v1.vals v2.vals];
0007
0008     v = unit.simplify(v);
0009
0010 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

plus

PURPOSE ^

SYNOPSIS ^

function v = plus(v1,v2)

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#)

This function is called by:

SOURCE CODE ^

```
0001
0002 function v = plus(v1,v2)
0003
0004     if ~strcmp(char(v1), char(v2))
0005         error('### Can''t add different units');
0006     end
0007
0008     v = v1;
0009
0010 end
```


power

PURPOSE ^

SYNOPSIS ^

```
function v = power(vi, exp)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

 [copy](#) COPY Make copy of unit objects depending of the second input

This function is called by:

SOURCE CODE ^

```
0001
0002 function v = power(vi, exp)
0003     v = copy(vi,1);
0004     if isa(exp, 'unit')
0005         exp = exp.exps;
0006     end
0007     if ~isempty(exp)
0008         v.exps = v.exps*exp;
0009         v.vals = v.vals.^exp;
0010     end
0011 end
```


rdivide

PURPOSE ^

SYNOPSIS ^

`function v = rdivide(v1,v2)`

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

 [unit](#) MINFO a helper class for implementing units in LTPDA.

This function is called by:

SOURCE CODE ^

```
0001
0002 function v = rdivide(v1,v2)
0003     v = unit;
0004     v.strs = [v1.strs v2.strs];
0005     v.exps = [v1.exps -1*v2.exps];
0006     v.vals = [v1.vals v2.vals];
0007
0008     v = unit.simplify(v);
0009
0010 end
```


sqrt

PURPOSE ^

SYNOPSIS ^

```
function v = sqrt(vi)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [copy](#) COPY Make copy of unit objects depending of the second input
-  [sqrt](#)

This function is called by:

-  [sqrt](#)

SOURCE CODE ^

```
0001
0002 function v = sqrt(vi)
0003     v = copy(vi, 1);
0004     v.exps = v.exps*0.5;
0005     v.vals = sqrt(v.vals);
0006 end
```


string

PURPOSE ^

STRING converts a unit object to a command string which will recreate the unit object.

SYNOPSIS ^

function varargout = string(varargin)

DESCRIPTION ^

```
STRING converts a unit object to a command string which will recreate the unit object.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: STRING converts a unit object to a command string which will
             recreate the unit object.

CALL:       cmd = string(u)

M-FILE INFO: Get information about this methods by calling
             >> unit.getInfo('string')

             Get information about a specified set-plist by calling:
             >> unit.getInfo('string', 'set')

VERSION:    $Id: string.m,v 1.1 2008/09/05 14:10:15 hewitson Exp $

HISTORY:    29-03-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [char](#)

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(\)](#)

SOURCE CODE ^

```
0001 % STRING converts a unit object to a command string which will recreate the unit
0002 % object.
0003 %
0004 % DESCRIPTION: STRING converts a unit object to a command string which will
0005 %               recreate the unit object.
0006 %
0007 % CALL:       cmd = string(u)
0008 %
0009 % M-FILE INFO: Get information about this methods by calling
0010 %             >> unit.getInfo('string')
0011 %
0012 %             Get information about a specified set-plist by calling:
0013 %             >> unit.getInfo('string', 'set')
0014 %
0015 % VERSION:    $Id: string.m,v 1.1 2008/09/05 14:10:15 hewitson Exp $
```

```
0016 %
0017 % HISTORY:      29-03-2007 M Hewitson
0018 %              Creation
0019 %
0020 %%%%%%%%%%%%%%
0021
0022 function varargout = string(varargin)
0023
0024 %%% Check if this is a call for parameters
0025 if utils.helper.isinfocall(varargin{:})
0026     varargout{1} = getInfo(varargin{3});
0027     return
0028 end
0029
0030 objs = utils.helper.collect_objects(varargin(:), 'unit');
0031
0032 pstr = '[';
0033
0034 for ii = 1:numel(objs)
0035     pstr = [pstr 'unit('];
0036     pstr = [pstr strrep(strrep(char(objs(ii)), '[', ' '), ']', ' ')];
0037     pstr = [pstr '')'];
0038
0039 end
0040
0041 pstr = [pstr '']];
0042
0043 varargout{1} = pstr;
0044 end
0045
0046 %%%%%%%%%%%%%%
0047 %              Local Functions
0048 %%%%%%%%%%%%%%
0049
0050 %%%%%%%%%%%%%%
0051 %
0052 % FUNCTION:      getInfo
0053 %
0054 % DESCRIPTION:   Get Info Object
0055 %
0056 % HISTORY:      11-07-07 M Hewitson
0057 %              Creation.
0058 %
0059 %%%%%%%%%%%%%%
0060
0061 function ii = getInfo(varargin)
0062     if nargin == 1 && strcmpi(varargin{1}, 'None')
0063         sets = {};
0064         pl = [];
0065     else
0066         sets = {'Default'};
0067         pl = getDefaultPlist;
0068     end
0069     % Build info object
0070     ii = minfo(mfilename, 'unit', '', utils.const.categories.helper, '$Id: string.m,v 1.1
2008/09/05 14:10:15 hewitson Exp $', sets, pl);
0071 end
0072
0073 %%%%%%%%%%%%%%
0074 %
0075 % FUNCTION:      getDefaultPlist
0076 %
0077 % DESCRIPTION:   Get Default Plist
0078 %
0079 % HISTORY:      11-07-07 M Hewitson
0080 %              Creation.
0081 %
0082 %%%%%%%%%%%%%%
0083
0084 function plo = getDefaultPlist()
0085     plo = plist();
0086 end
0087
```

[Home](#) > [classes](#) > [@unit](#) > times.m

times

PURPOSE [^]

SYNOPSIS [^]

```
function v = times(v1,v2)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [unit](#) MINFO a helper class for implementing units in LTPDA.

This function is called by:

SOURCE CODE [^]

```
0001
0002 function v = times(v1,v2)
0003     v = unit;
0004     v.strs = [v1.strs v2.strs];
0005     v.exps = [v1.exps v2.exps];
0006     v.vals = [v1.vals v2.vals];
0007
0008     v = unit.simplify(v);
0009
0010 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

unit

PURPOSE ^

MINFO a helper class for implementing units in LTPDA.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
MINFO a helper class for implementing units in LTPDA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

MINFO a helper class for implementing units in LTPDA.

>> u = unit(str);

Inputs:
  str - a string describing the units. All SI prefixes are
        supported. Exponents on the units are supported. Multiple
        units can be entered by separating them by whitespaces.

Outputs:
  ii - an unit object

EXAMPLES:

1) Create a simple unit:
  >> u = unit('m');

2) With an exponent:
  >> u = unit('m^3');

3) With a prefix:
  >> u = unit('pm^2');

4) Multiple units:
  >> u = unit('m s^-2 kg');

Supported Prefixes: unit.supportedPrefixes
Supported Units:    unit.supportedUnits

M Hewitson 05-09-08

VERSION: $Id: unit.m,v 1.3 2008/09/08 08:28:27 hewitson Exp $

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%classdef unit
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [unit](#) MINFO a helper class for implementing units in LTPDA.

This function is called by:

-  [copy](#) COPY Make copy of unit objects depending of the second input
-  [mrdivide](#)
-  [mtimes](#)

-  `rdivide`
-  `times`
-  `unit` MINFO a helper class for implementing units in LTPDA.

SUBFUNCTIONS ^

-  `function u = unit(varargin)`
-  `function [us, exp, val] = parse(ustr)`
-  `function v = simplify(v)`
-  `function val = prefix2val(p)`
-  `function out = VEROUT()`
-  `function ii = getInfo(varargin)`
-  `function out = SETS()`
-  `function out = getDefaultPlist(set)`
-  `function pfxs = supportedPrefixes`
-  `function pfxs = supportedUnits`

SOURCE CODE ^

```
0001 % MINFO a helper class for implementing units in LTPDA.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % MINFO a helper class for implementing units in LTPDA.
0005 %
0006 % >> u = unit(str);
0007 %
0008 % Inputs:
0009 % str - a string describing the units. All SI prefixes are
0010 % supported. Exponents on the units are supported. Multiple
0011 % units can be entered by separating them by whitespaces.
0012 %
0013 % Outputs:
0014 % ii - an unit object
0015 %
0016 % EXAMPLES:
0017 %
0018 % 1) Create a simple unit:
0019 %
0020 % >> u = unit('m');
0021 %
0022 % 2) With an exponent:
0023 %
0024 % >> u = unit('m^3');
0025 %
0026 % 3) With a prefix:
0027 %
0028 % >> u = unit('pm^2');
0029 %
0030 % 4) Multiple units:
0031 %
0032 % >> u = unit('m s^-2 kg');
0033 %
0034 % Supported Prefixes: unit.supportedPrefixes
0035 % Supported Units: unit.supportedUnits
0036 %
0037 %
0038 %
0039 % M Hewitson 05-09-08
0040 %
0041 % VERSION: $Id: unit.m,v 1.3 2008/09/08 08:28:27 hewitson Exp $
0042 %
0043 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%classdef
0044 unit
0045 classdef unit < ltpda_nuo
0046
0047 %-----
0048 %- Private properties
0049 %-----
0050 properties (SetAccess = protected)
0051     strs = {};
0052     exps = [];
0053     vals = [];
```



```

0054     version = '$Id: unit.m,v 1.3 2008/09/08 08:28:27 hewitson Exp $';
0055 end
0056
0057
0058 %-----
0059 %- Public methods
0060 %-----
0061 methods
0062
0063 %-----
0064 %- Constructor
0065 %-----
0066 function u = unit(varargin)
0067
0068     switch nargin
0069     case 0
0070         % Empty constructor
0071     case 1
0072         % String input
0073
0074         ustr = varargin{1};
0075
0076         if ischar(ustr)
0077             if ~isempty(ustr)
0078                 ustr = strrep(strrep(ustr, '[', ' '), ']', ' ');
0079
0080                 % split on whitespace
0081                 tks = regexp(strtrim(ustr), '\s+', 'split');
0082                 % combine each unit
0083                 for j=1:numel(tks)
0084                     % Parse string
0085                     [us, exp, val] = unit.parse(tks{j});
0086                     u2.strs = {us};
0087                     u2.exps = exp;
0088                     u2.vals = val;
0089                     u = u*u2;
0090                 end
0091             end
0092         elseif isstruct(ustr)
0093             u.strs = ustr.strs;
0094             u.exps = ustr.exps;
0095             u.vals = ustr.vals;
0096             u.version = ustr.version;
0097         elseif isa(ustr, 'unit')
0098             u = ustr;
0099         else
0100             error('### Unknown single argument constructor');
0101         end
0102     end
0103 end
0104
0105 end % End constructor
0106
0107 end % End public methods
0108
0109 %-----
0110 %- Private Static methods
0111 %-----
0112 methods (Static=true, Access=private)
0113
0114 %-----
0115 %- Parse a unit definition string
0116 %-----
0117 function [us, exp, val] = parse(ustr)
0118
0119 % get exp
0120 [s,t] = strtok(ustr, '^');
0121 if ~isempty(t)
0122     % drop any (
0123     t = strrep(t(2:end), '(', '');
0124     t = strrep(t, ')', '');
0125     exp = eval(t);
0126 else
0127     exp = 1;
0128 end
0129
0130 % check for prefix
0131 if ismember(s(1), unit.supportedPrefixes) && length(s) > 1 && ismember(s(2:end),
unit.supportedUnits)
0132     sp = s(1);
0133     val = unit.prefix2val(sp);
0134     sm = s(2:end);
0135 else
0136     val = 1;
0137 end
0138

```

```

0139         sp = '';
0140         sm = s;
0141     end
0142
0143     % Check unit
0144     if ~ismember(sm, unit.supportedUnits)
0145         error('### Unsupported unit: [' sp ']' sm]);
0146     end
0147
0148     % set unit string
0149     us = [sp sm];
0150
0151 end % End parse
0152
0153 %-----
0154 %- Simplify a set of units
0155 %-----
0156 function v = simplify(v)
0157
0158     for j=1:numel(v.strs)
0159         for k=j+1:numel(v.strs)
0160             if strcmp(v.strs{j}, v.strs{k})
0161                 v.exps(j) = v.exps(j) + v.exps(k);
0162                 v.strs{k} = '';
0163                 v.exps(k) = 0;
0164             end
0165         end
0166     end
0167
0168     idx = find(v.exps~=0);
0169     v.strs = v.strs(idx);
0170     v.exps = v.exps(idx);
0171     v.vals = v.vals(idx);
0172
0173 end
0174
0175 %-----
0176 %- Get the value associated with a prefix
0177 %-----
0178 function val = prefix2val(p)
0179     pfxs = unit.supportedPrefixes;
0180     idx = find(strcmp(p, pfxs));
0181     val = 1e-24*10^(3*(idx-1));
0182 end
0183
0184
0185 end % End static private methods
0186
0187 %-----
0188 %- Public static methods
0189 %-----
0190 methods (Static=true)
0191
0192     function out = VEROUT()
0193         out = '$Id: unit.m,v 1.3 2008/09/08 08:28:27 hewitson Exp $';
0194     end
0195
0196     function ii = getInfo(varargin)
0197         ii = utils.helper.generic_getInfo(varargin{:}, 'unit');
0198     end
0199
0200     function out = SETS()
0201         out = {'Default'};
0202     end
0203
0204     function out = getDefaultPlist(set)
0205         switch set
0206             case 'Default'
0207                 out = plist();
0208             otherwise
0209                 out = plist();
0210         end
0211     end
0212
0213
0214 %-----
0215 %- Return a list of supported prefixes
0216 %-----
0217 function pfxs = supportedPrefixes
0218     pfxs = {'y', 'z', 'a', 'f', 'p', 'n', 'u', 'm', 'c', ...
0219         'h', 'k', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y'};
0220 end
0221
0222 %-----
0223 %- Return a list of supported units
0224 %-----

```

```
0225     function pfxs = supportedUnits
0226         pfxs = {'m', 'kg', 's', 'A', 'K', 'mol', 'cd', 'i', 'V', 'F', ...
0227             'rad', 'sr', 'Hz', 'N', 'Pa', 'J', 'W', 'C', 'i', 'V', 'F', ...
0228             'Ohm', 'S', 'Wb', 'T', 'H', 'degC', ...
0229             'i', 'I', 'Count', 'A/U', 'arb'};
0230     end
0231 end
0232 end
0233 end
```

update_struct

PURPOSE ^

UPDATE_STRUCT update the input structure to the current ltpda version

SYNOPSIS ^

function varargout = update_struct(varargin)

DESCRIPTION ^

UPDATE_STRUCT update the input structure to the current ltpda version
%%%

FUNCTION: update_struct

DESCRIPTION: UPDATE_STRUCT update the input structure to the current
 ltpda version

CALL: obj_struct = update_struct(obj_struct, version_str);

VERSION: \$Id: update_struct.m,v 1.1 2008/09/05 14:10:15 hewitson Exp \$

HISTORY: 07-07-2008 Diepholz
 Creation

%%%

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % UPDATE_STRUCT update the input structure to the current ltpda version
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % FUNCTION:        update_struct
0005 %
0006 % DESCRIPTION: UPDATE_STRUCT update the input structure to the current
0007 %                ltpda version
0008 %
0009 % CALL:            obj_struct = update_struct(obj_struct, version_str);
0010 %
0011 % VERSION:        $Id: update_struct.m,v 1.1 2008/09/05 14:10:15 hewitson Exp $
0012 %
0013 % HISTORY:        07-07-2008 Diepholz
0014 %                Creation
0015 %
0016 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0017 %
0018 function varargout = update_struct(varargin)
0019 %
0020 obj_struct = varargin{1};
0021 struct_ver = varargin{2};
0022 %
0023 % get the version of the current toolbox
0024 tbx_ver = strtok(getappdata(0, 'ltpda_version'));
0025 %
0026 % get only the version string without the MATLAB version
0027 struct_ver = strtok(struct_ver);
0028 %
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    Update version '1.0 -> 1.1'    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030 %
0031 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
```

```

0032     strcmp(struct_ver, '1.0')
0033
0034     try
0035
0036     catch ME
0037         disp(varargin{1});
0038         throw(addCause(ME, MException('MATLAB:LTPDA', '### The struct (unit) above is not
from version 1.0')));
0039     end
0040
0041     % Set the struct_ver to '1.1'
0042     struct_ver = '1.9.1';
0043
0044 end
0045
0046 %%%%%%%%%% Update version '1.9.1' ->'1.9.2' %%%%%%%%%%
0047
0048 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0049     strcmp(struct_ver, '1.9.1')
0050
0051     % Set the struct_ver to '1.9.2'
0052     struct_ver = '1.9.2';
0053
0054 end
0055
0056 %%%%%%%%%% Update version '1.2' ->'1.3' %%%%%%%%%%
0057
0058 if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
0059     strcmp(struct_ver, '1.2')
0060
0061     % Set the struct_ver to '1.3'
0062     struct_ver = '1.3';
0063
0064 end
0065
0066 vararginout{1} = obj_struct;
0067

```

Contents

PURPOSE ^

%% class: xydata %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: xydata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help xydata/copy">xydata/copy</a>           - Make copy of xydata objects
depending of the second input
    <a href="matlab:help xydata/display">xydata/display</a>     - overloads display
functionality for xydata objects.
    <a href="matlab:help xydata/update_struct">xydata/update_struct</a> - update the input
structure to the current ltpda version
    <a href="matlab:help xydata/xydata">xydata/xydata</a>       - X-Y data object class
constructor.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% class: xyzdata %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% class: xyzdata %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help xyzdata/char">xyzdata/char</a>           - convert a ltpda_data-
object into a string.
    <a href="matlab:help xyzdata/copy">xyzdata/copy</a>           - Make copy of xyzdata
objects depending of the second input
    <a href="matlab:help xyzdata/display">xyzdata/display</a>     - overloads display
functionality for xyzdata objects.
    <a href="matlab:help xyzdata/update_struct">xyzdata/update_struct</a> - update the input
structure to the current ltpda version
    <a href="matlab:help xyzdata/xyzdata">xyzdata/xyzdata</a>     - XZYDATA X-Y-Z data
object class constructor.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% path: m/etc %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/etc %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    <a href="matlab:help m/etc/ao_fcn_template">m/etc/ao_fcn_template</a>      - F00
description for function 'foo' in one line. Necessary for lookfor functionality.
    <a href="matlab:help m/etc/ltpda_startup">m/etc/ltpda_startup</a>          - This is the
startup file for ltpda. It should be run once in the MATLAB
    <a href="matlab:help m/etc/ltpda_user_fcn_template">m/etc/ltpda_user_fcn_template</a> -
LTPDA_F00 is the standard function template for user functions
    m/etc/ltpda_versions              - (No help available)
    <a href="matlab:help m/etc/method_fcn_template">m/etc/method_fcn_template</a>      - F00 is
the standard function template for class methods.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ao_fcn_template

PURPOSE ^

FOO description for function 'foo' in one line. Necessary for lookfor functionality.

SYNOPSIS ^

function varargout = foo(varargin)

DESCRIPTION ^

```
FOO description for function 'foo' in one line. Necessary for lookfor functionality.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: FOO detailed description for the function 'foo'

CALL:
    foo(a)
    [a, b] = foo(a,pl)

INPUTS:
    a - analysis object(s)
    pl - parameter list(s)

OUTPUTS:
    a - ???
    b - ???

M-FILE INFO: Get information about this methods by calling
    >> ao.getInfo('foo')

    Get information about a specified set-plist by calling:
    >> ao.getInfo('foo', 'set')

VERSION: $Id: ao_fcn_template.m,v 1.14 2008/08/07 15:37:23 ingo Exp $

HISTORY: dd-mm-yyyy M Hewitson
    Creation

SEE ALSO: ao/cos, ao/tan, ao/asin, acos, atan
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function ii = getInfo\(varargin\)](#)
-  [function plo = getDefaultPlist\(set\)](#)

SOURCE CODE ^

```
0001 % FOO description for function 'foo' in one line. Necessary for lookfor functionality.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: FOO detailed description for the function 'foo'
0005 %
0006 % CALL:
0007 %     foo(a)
0008 %     [a, b] = foo(a,pl)
0009 %
0010 % INPUTS:
0011 %     a - analysis object(s)
0012 %     pl - parameter list(s)
```

```

0012 % OUTPUTS:      a - ???
0013 %              b - ???
0014 %
0015 % M-FILE INFO:  Get information about this methods by calling
0016 %              >> ao.getInfo('foo')
0017 %
0018 %              Get information about a specified set-plist by calling:
0019 %              >> ao.getInfo('foo', 'set')
0020 %
0021 % VERSION:      $Id: ao_fcn_template.m,v 1.14 2008/08/07 15:37:23 ingo Exp $
0022 %
0023 % HISTORY:      dd-mm-yyyy M Hewitson
0024 %              Creation
0025 %
0026 % SEE ALSO:      ao/cos, ao/tan, ao/asin, acos, atan
0027 %
0028 %%%%%%%%%%%%%%
0029
0030 function varargout = foo(varargin)
0031
0032 %%% Check if this is a call for parameters
0033 if utils.helper.isinfocall(varargin{:})
0034     varargout{1} = getInfo(varargin{3});
0035     return
0036 end
0037
0038 %%% Collect input variable names
0039 in_names = cell(size(varargin));
0040 for ii = 1:nargin,in_names{ii} = inputname(ii);end
0041
0042 %%% Collect all AOs
0043 [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
0044 pli             = utils.helper.collect_objects(varargin(:), 'plist', in_names);
0045
0046 %%% Decide on a deep copy or a modify
0047 %%% REMARK: If you create a new AO (call the constructor) then
0048 %%%          it is not necessary to copy the input-AOs !!!!!!!!!!!!!!!!!!!!!!!
0049 bs = copy(as, nargout);
0050
0051 %%% Combine plists
0052 pl = combine(pli, getDefaultPlist);
0053
0054 %%% go through analysis objects
0055 for kk = 1: numel(bs)
0056
0057     %%%%%%%%% some calculations %%%%%%%%%
0058     bs(kk).data.setY(some_new_data);
0059     bs(kk).data.setFs(new_fs);
0060     bs(kk).data.setXunits(new_xunits);
0061
0062     %%% Set Name
0063     bs(kk).setName('new name', 'internal');
0064     %%% A better way to set the name
0065     bs(kk).name = 'new name';
0066
0067     %%% Add History
0068     bs(kk).addHistory(getInfo('Special set or all sets? You decide.'), pli,
0069     ao_invars(kk), bs(kk).hist);
0070 end
0071
0072 %%% Prepare the output
0073 varargout{1} = bs;
0074
0075 end
0076
0077 %%%%%%%%%%%%%%
0078 %              Local Functions %
0079 %%%%%%%%%%%%%%
0080
0081 %%%%%%%%%%%%%%
0082 %
0083 % FUNCTION:      getInfo
0084 %
0085 % DESCRIPTION:  Get Info Object
0086 %
0087 % HISTORY:      11-07-07 M Hewitson
0088 %              Creation.
0089 %
0090 %%%%%%%%%%%%%%
0091
0092 function ii = getInfo(varargin)
0093     if nargin == 1 && strcmpi(varargin{1}, 'None')
0094         sets = {};
0095         pls  = [];
0096     elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})

```

```
0097     sets{1} = varargin{1};
0098     pls = getDefaultPlist(sets{1});
0099 else
0100     sets = {'set1', 'set2', 'set3'};
0101     pls = [];
0102     for kk=1:numel(sets)
0103         pls = [pls getDefaultPlist(sets{kk})];
0104     end
0105 end
0106 % Build info object
0107 ii = minfo(mfilename, 'CLASS', '', 'CATEGORY', '$Id: ao_fcn_template.m,v 1.14
2008/08/07 15:37:23 ingo Exp $', sets, pls);
0108 end
0109
0110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0111 %
0112 % FUNCTION:      getDefaultPlist
0113 %
0114 % DESCRIPTION:  Get Default Plist
0115 %
0116 % HISTORY:      11-07-07 M Hewitson
0117 %                Creation.
0118 %
0119 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0120
0121 function plo = getDefaultPlist(set)
0122 switch set
0123 case 'set1'
0124     plo = plist();
0125 case 'set2'
0126     plo = plist('a', 1, 'b', 2);
0127 case 'set3'
0128     plo = plist('key', 'val');
0129 otherwise
0130     error('### Unknown default plist for the set [%s]', set)
0131 end
0132 end
0133
```

ltpda_startup

PURPOSE ^

This is the startup file for ltpda. It should be run once in the MATLAB

SYNOPSIS ^

function ltpda_startup

DESCRIPTION ^

This is the startup file for ltpda. It should be run once in the MATLAB session before using any features of ltpda. The best way to ensure this is to create a file called startup.m and put this somewhere in your MATLAB path. In this file you should have the command 'ltpda_startup'.

M Hewitson 16-03-07

\$Id: ltpda_startup.m,v 1.44 2008/09/04 17:27:13 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_startup
0002
0003 % This is the startup file for ltpda. It should be run once in the MATLAB
0004 % session before using any features of ltpda. The best way to ensure this
0005 % is to create a file called startup.m and put this somewhere in your
0006 % MATLAB path. In this file you should have the command 'ltpda_startup'.
0007 %
0008 % M Hewitson 16-03-07
0009 %
0010 % $Id: ltpda_startup.m,v 1.44 2008/09/04 17:27:13 hewitson Exp $
0011 %
0012 %
0013 %-----
0014 % User Configurable parameters
0015 %
0016 %
0017 %
0018 %-----
0019 % Terminal output from LTPDA
0020 %
0021 VERBOSE_LEVEL = 1; % Choose the level of terminal output, higher numbers
0022 % means more output; 0 is no output.
0023 %
0024 %-----
0025 % Settings for SSM models
0026 %
0027 %
0028 % Give a path or cell-array of paths
0029 SSM_MODELS = '/Users/hewitson/working/ltp/ltp_ssm_models';
0030 %
0031 %-----
0032 % Print and plot settings
0033 %
0034 USE_LTPDA_PRINT_SETTINGS = 'Yes'; % Choose LTPDA Print setup
0035 USE_LTPDA_PLOT_SETTINGS = 'Yes'; % Choose LTPDA Plot setup
0036 %
0037 %-----
0038 % Various GUI settings
0039 %
```

Description of ltpda_startup

```

0040 % A list of repositories presented to the user
0042 REPO_GUI_SERVERLIST = {'localhost', '130.75.117.67', '130.75.117.61'};
0043
0044 % Font sizes
0045 GUI_FONTSIZE          = 12;          % Choose GUI font size
0046 REPO_GUI_FONTSIZE     = 10;          % Choose Repository GUI font size
0047 PASSWD_WIN_FONTSIZE   = 12;          % Choose login/passw dialog window font size
0048
0049 % Working directories for the main LTPDA GUI
0050 DEFAULT_DATA_DIR      = 'C:\data';
0051 DEFAULT_ANALYSES_DIR  = 'C:\analyses';
0052
0053
0054 %-----
0055 % Misc settings
0056 %
0057
0058 WRAP_STRINGS           = 8;           % Wrap strings at this value
0059 WRAP_LEGEND_STRINGS    = 10;          % Wrap legend strings at this value
0060 XML_SET_SIZE           = 50000;       % Set the maximum line length for writing data
0061 sets to XML
0062
0063 %-----
0064 % Time settings
0065 %
0066
0067 TIMEZONE               = 'UTC';       % Users timezone
0068                                     % To get valid IDs use:
0069                                     % java.util.TimeZone.getAvailableIDs
0070 TIME_FORMAT_STR        = 'yyyy-mm-dd HH:MM:SS.FFF'; % Users time format.
0071
0072
0073
0074 %-----
0075 %-----
0076 %-----
0077 %-----
0078 %-----
0079 %-----
0080 %-----
0081 %-----
0082 %-----
0083 %-----
0084 %-----
0085 %-----
0086 %-----
0087 %-----
0088 %-----
0089 %-----
0090 %-----
0091 %-----
0092 %-----
0093 %-----
0094 %-----
0095 %-----
0096 %-----
0097 %-----
0098 %-----
0099 %-----
0100 %-----
0101 %-----
0102
0103
0104 %-----
0105 %-----
0106 % NO NEED TO EDIT BELOW HERE
0107 %-----
0108 %-----
0109
0110 v = ver('LTPDA');
0111
0112 logo = {...
0113
0114
0115
0116
0117
0118
0119
0120
0121
0122
0123
0124

```



```

0125 '      \\\ \      //      \\\ \      '
0126 '      - - - - -      '
0127 '      * *      '
0128 '      * * * *      '
0129 };
0130
0131 l1 = '+-----+';
0132 ll = length(l1);
0133
0134 disp(l1);
0135 disp('|')
0136 for j=1:length(logo)
0137     disp([utils.prog.strpad(sprintf('%s ', char(logo{j})), ll-1) '|']);
0138 end
0139 disp([utils.prog.strpad(' ', ll-1) '|'])
0140 disp([utils.prog.strpad(sprintf('Welcome to %s', v.Name), ll-1) '|'])
0141 disp([utils.prog.strpad(' ', ll-1) '|'])
0142 disp([utils.prog.strpad(sprintf('Version: %s', v.Version), ll-1) '|'])
0143 disp([utils.prog.strpad(sprintf('Release: %s', v.Release), ll-1) '|'])
0144 disp([utils.prog.strpad(sprintf('Date: %s', v.Date), ll-1) '|'])
0145 disp([utils.prog.strpad(' ', ll-1) '|'])
0146 disp(l1);
0147
0148 %-----
0149 % set page properties for printing
0150 if strcmpi(USE_LTPDA_PRINT_SETTINGS, 'YES')
0151     set(0, 'DefaultFigurePaperOrientation', 'portrait');
0152     set(0, 'DefaultFigurePaperType', 'A4');
0153     set(0, 'DefaultFigurePaperUnits', 'inches');
0154 end
0155
0156 %-----
0157 % Plot settings
0158 if strcmpi(USE_LTPDA_PLOT_SETTINGS, 'YES')
0159     set(0, 'DefaultAxesFontSize', 14);
0160     set(0, 'DefaultAxesFontWeight', 'bold');
0161     set(0, 'DefaultAxesLineWidth', 1);
0162     set(0, 'DefaultLineLineWidth', 2);
0163     set(0, 'defaultlinemarkersize', 10);
0164     set(0, 'DefaultAxesXColor', [0.2 0.2 0.8]);
0165     set(0, 'DefaultAxesYColor', [0.2 0.2 0.8]);
0166     set(0, 'DefaultAxesGridLineStyle', '-');
0167     set(0, 'DefaultAxesMinorGridLineStyle', '-');
0168     set(0, 'defaultfigurenumbertitle', 'on');
0169     set(0, 'DefaultFigureColor', 'w');
0170     set(0, 'DefaultFigurePosition', [100 100 800 600]);
0171     set(0, 'DefaultFigurePaperPositionMode', 'auto');
0172
0173 end
0174
0175 %-----
0176 % format of numbers on MATLAB terminal
0177 format long g
0178
0179 % -----
0180 % MySQL LTPDA Repository Server Settings
0181 %
0182 if exist('REPOUSER', 'var') == 0
0183     REPOUSER = '';
0184 end
0185 if exist('REPOPASSWD', 'var') == 0
0186     REPOPASSWD = '';
0187 end
0188
0189 DBDRIVER = 'com.mysql.jdbc.Driver'; % Set LTPDA Repository database driver
0190
0191 setappdata(0, 'ltpda_repo_user', REPOUSER);
0192 setappdata(0, 'ltpda_repo_pass', REPOPASSWD);
0193 setappdata(0, 'mysql_driver', DBDRIVER);
0194 setappdata(0, 'ltpda_repo_servers', REPO_GUI_SERVERLIST);
0195 setappdata(0, 'ltpda_gui_fontsize', GUI_FONTSIZE);
0196 setappdata(0, 'ltpda_repo_gui_fontsize', REPO_GUI_FONTSIZE);
0197 setappdata(0, 'ltpda_passwd_win_fontsize', PASSWD_WIN_FONTSIZE);
0198
0199 % add mysql JDBC driver to the java path
0200 mysqldriver = [fileparts(which('ltpda_startup')) '../classes/+utils/@mysql/mysql-
connector-java-5.1.6-bin.jar'];
0201 javaaddpath(mysqldriver);
0202
0203 %-----
0204 % set default directories to load the data from and save the model in
0205
0206 setappdata(0, 'DefaultDataDirectory', DEFAULT_DATA_DIR);
0207 setappdata(0, 'DefaultAnalysesDirectory', DEFAULT_ANALYSES_DIR);
0208
0209

```

```

0210 try
0211     setdbprefs('NullStringRead','Null');
0212     setdbprefs('NullNumberRead','NaN');
0213 catch
0214     warning('!!! No database toolbox is installed');
0215     warning('!!! You won't be able to work with an LTPDA Repository');
0216 end
0217
0218 % -----
0219 % General Variables
0220 setappdata(0, 'wrapstringat', WRAP_STRINGS); % wrap strings at this length for
0221                                             % history plot nodes
0222 setappdata(0, 'wraplegendstringat', WRAP_LEGEND_STRINGS); % wrap strings at this length
for legends
0223 setappdata(0, 'xmlsetsize', XML_SET_SIZE); % Max size of an xml data set <Set></Set>
0224
0225 setappdata(0, 'ltpda_default_plot_colors', {[0.8 0.2 0.2], ...
0226                                             [0.2 0.2 0.8], ...
0227                                             [0.2 0.9 0.2], ...
0228                                             [0.37 0.9 0.83], ...
0229                                             [0.888 0.163 0.9], ...
0230                                             [0 0 0], ...
0231                                             [0 207 255]/255, ...
0232                                             [255 128 0]/255, ...
0233                                             [143 0 0]/255, ...
0234                                             [255 207 0]/255, ...
0235                                             [0.9 0.266 0.593]}]);
0236
0237 setappdata(0, 'timezone', TIMEZONE);
0238 setappdata(0, 'time_format_str', TIME_FORMAT_STR);
0239
0240
0241 % -----
0242 % Version Variables
0243
0244 vs = ver;
0245 for j=1:length(vs)
0246     v = vs(j);
0247     switch v.Name
0248     case 'MATLAB'
0249         matlab_version = [v.Version ' ' v.Release];
0250     case 'Signal Processing Toolbox'
0251         sigproc_version = [v.Version ' ' v.Release];
0252     case 'Symbolic Math Toolbox'
0253         ltpda_symbolic_math = [v.Version ' ' v.Release];
0254     case 'LTPDA Toolbox'
0255         ltpda_version = [v.Version ' ' v.Release];
0256     end
0257 end
0258
0259 setappdata(0, 'matlab_version', matlab_version);
0260 setappdata(0, 'sigproc_version', sigproc_version);
0261 setappdata(0, 'ltpda_symbolic_math', ltpda_symbolic_math);
0262 setappdata(0, 'ltpda_version', ltpda_version);
0263 setappdata(0, 'ltpda_required_matlab_version', '7.6');
0264
0265 % -----
0266 % Other settings
0267
0268 setappdata(0, 'ltpda_default_spectral_window', specwin('BH92', 10));
0269 setappdata(0, 'ltpda_verbose_level', VERBOSE_LEVEL);
0270 setappdata(0, 'ltpda_ssm_model_paths', SSM_MODELS);
0271
0272 if ischar(SSM_MODELS)
0273     SSM_MODELS = {SSM_MODELS};
0274 end
0275 for j=1:numel(SSM_MODELS)
0276     addpath(SSM_MODELS{j});
0277 end
0278
0279 % -----
0280 % import some things
0281 import utils.const.*
0282
0283 % -----
0284 % Launch some things
0285
0286 ltpdalauncher;
0287
0288
0289
0290 % END

```

ltpda_user_fcn_template

PURPOSE ^

LTPDA_FOO is the standard function template for user functions

SYNOPSIS ^

function varargout = ltpda_foo(varargin)

DESCRIPTION ^

```
LTPDA_FOO is the standard function template for user functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_FOO is the standard function template for LTPDA user
functions
CALL:
      ltpda_foo(a)
      [a, b] = ltpda_foo(a,pl)
INPUTS:
      a - analysis object(s)
      pl - parameter list(s)
      ...
OUTPUTS:
      a - ???
      b - ???
PARAMETERS: (Add these parameters to the default parameter list)
      A: - parameter description
      X: - parameter description
      DESCRIPTION: - user defined routine description
VERSION:      $Id: ltpda_user_fcn_template.m,v 1.4 2008/03/28 11:16:46 mauro Exp $
HISTORY:      dd-mm-yyyy M Hewitson
               Creation
The following call returns a parameter list object that contains the
default parameter values:
>> pl = ltpda_foo('Params')
The following call returns a string that contains the routine CVS version:
>> version = ltpda_foo('Version')
The following call returns a string that contains the routine category:
>> category = ltpda_foo('Category')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

 [function plo = getDefaultPL\(\)](#)

SOURCE CODE ^

```

0001 function varargout = ltpda_foo(varargin)
0002 % LTPDA_FOO is the standard function template for user functions
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_FOO is the standard function template for LTPDA user
0007 % functions
0008 %
0009 % CALL:          ltpda_foo(a)
0010 %               [a, b] = ltpda_foo(a,pl)
0011 %
0012 % INPUTS:        a - analysis object(s)
0013 %               pl - parameter list(s)
0014 %               ...
0015 %
0016 % OUTPUTS:       a - ???
0017 %               b - ???
0018 %
0019 % PARAMETERS:    (Add these parameters to the default parameter list)
0020 %   A: - parameter description
0021 %   X: - parameter description
0022 %   DESCRIPTION: - user defined routine description
0023 %
0024 % VERSION:       $Id: ltpda_user_fcn_template.m,v 1.4 2008/03/28 11:16:46 mauro Exp $
0025 %
0026 % HISTORY:       dd-mm-yyyy M Hewitson
0027 %               Creation
0028 %
0029 % The following call returns a parameter list object that contains the
0030 % default parameter values:
0031 %
0032 % >> pl = ltpda_foo('Params')
0033 %
0034 % The following call returns a string that contains the routine CVS version:
0035 %
0036 % >> version = ltpda_foo('Version')
0037 %
0038 % The following call returns a string that contains the routine category:
0039 %
0040 % >> category = ltpda_foo('Category')
0041 %
0042 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0043 %
0044 %
0045 %
0046 %% ----- Standard history variables
0047
0048 ALGONAME = mfilename;
0049 VERSION  = '$Id: ltpda_user_fcn_template.m,v 1.4 2008/03/28 11:16:46 mauro Exp $';
0050 CATEGORY = 'Signal Processing';
0051
0052
0053 %% Check if this is a call for parameters, the CVS version string
0054 % or the function category
0055 if nargin == 1 && ischar(varargin{1})
0056     in = char(varargin{1});
0057     if strcmp(in, 'Params')
0058         varargout{1} = getDefaultPL();
0059         return
0060     elseif strcmp(in, 'Version')
0061         varargout{1} = VERSION;
0062         return
0063     elseif strcmp(in, 'Category')
0064         varargout{1} = CATEGORY;
0065         return
0066     end
0067 end
0068
0069
0070 %% Collect input ao's, plist's and ao variable names
0071 in_names = {};
0072 for ii = 1:nargin
0073     in_names{end+1} = inputname(ii);
0074 end
0075
0076 [as, pl, invars] = collect_inputs(varargin, in_names);
0077
0078 % produce one parameter list
0079 if isa(pl, 'plist')
0080     pl = combine(pl, getDefaultPL());
0081 else
0082     pl = getDefaultPL();
0083 end
0084
0085
0086 %% Get parameters

```

```

0087 A = find(pl, 'A');
0088 X = find(pl, 'X');
0089
0090
0091 %% Initialize outputs
0092 bs = [];
0093
0094
0095 %% Go through analysis objects
0096 for j=1:numel(as)
0097     a = as(j);
0098
0099     % get data
0100     [x,y] = get_xy_values(a.data);
0101
0102     % Do some analysis
0103     y = 2*A * (y + X);
0104     x = 1:length(x);
0105
0106     %%----- Build output A0s
0107
0108     % make a new xydata object, for example
0109     xy = xydata(x, y);
0110     xy = set(xy, 'name', sprintf('ltpda_foo(%s)', a.data.name));
0111     xy = set(xy, 'xunits', a.data.xunits);
0112     xy = set(xy, 'yunits', 'Blah');
0113
0114     % make a new history object
0115     h = history(ALGONAME, VERSION, pl, a.hist);
0116     h = set(h, 'invars', invars);
0117
0118     % make output analysis object
0119     b = ao(xy, h);
0120
0121     % name, mfilename, description for this object
0122     b = setnh(b, 'name', sprintf('ltpda_foo(%s)', invars{j}), ...
0123         'description', find(pl, 'description'));
0124
0125     bs = [bs b];
0126
0127 end
0128
0129
0130 %% Reshape the ouput to the same size of the input
0131 varargout{1} = reshape(bs, size(as));
0132
0133
0134 %% %%%%%%%%%%%
0135 %
0136 % FUNCTION:      getDefaultPL
0137 %
0138 % DESCRIPTION:  Get default params
0139 %
0140 % HISTORY:      dd-mm-yyyy M Hewitson
0141 %               Creation
0142 %
0143 %%%%%%%%%%%
0144 function plo = getDefaultPL()
0145
0146 plo = plist('A', 10, ...
0147     'X', [1], ...
0148     'DESCRIPTION', 'LTPDA User Function Template');
0149
0150
0151 % END

```

ltpda_versions

PURPOSE ^

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 versions = {
0002     'SETS' 'ltpda_obj';
0003     'VEROUT' 'ltpda_obj';
0004     'addlistener' 'ltpda_obj';
0005     'char' 'ltpda_obj';
0006     'copy' 'ltpda_obj';
0007     'delete' 'ltpda_obj';
0008     'display' 'ltpda_obj';
0009     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/eq.m',
0010     'eq' '1.8' '2008/09/04' '15:29:30' 'ingo'
0011     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
0012     '08af28fc64b0b5c7d96638545132d050' 'ltpda_obj';
0013     'findobj' 'ltpda_obj';
0014     'findprop' 'ltpda_obj';
0015     'ge' 'ltpda_obj';
0016     'getDefaultPlist' 'ltpda_obj';
0017     'getInfo' 'ltpda_obj';
0018     'gt' 'ltpda_obj';
0019     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/isprop.m', 'isprop',
0020     '1.2' '2008/09/04' '15:29:30' 'ingo'
0021     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
0022     '61cf4eab96bd5818098a39bf62a429cb' 'ltpda_obj';
0023     'isvalid' 'ltpda_obj';
0024     'le' 'ltpda_obj';
0025     'lt' 'ltpda_obj';
0026     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/ltpda_obj.m',
0027     'ltpda_obj' '1.22' '2008/09/06' '12:24:27' 'hewitson'
0028     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
0029     '94dfb51dee1516944a68b270de6bd5aa' 'Constructor' 'ltpda_obj';
0030     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/ne.m',
0031     'ne' '1.5' '2008/09/04' '15:29:30' 'ingo'
0032     '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
0033     '01a6311c65819cae935a156c167afffe' 'ltpda_obj';
0034     'notify' 'ltpda_obj';
0035     'update_struct' 'ltpda_obj';
0036     'SETS' 'ltpda_nuo';
0037     'VEROUT' 'ltpda_nuo';
0038     'addlistener' 'ltpda_nuo';
0039     'char' 'ltpda_nuo';
0040     'copy' 'ltpda_nuo';
0041     'delete' 'ltpda_nuo';
0042     'display' 'ltpda_nuo';
0043     'eq' 'ltpda_nuo';
0044     'findobj' 'ltpda_nuo';
0045     'findprop' 'ltpda_nuo';
0046     'ge' 'ltpda_nuo';
0047     'getDefaultPlist' 'ltpda_nuo';
0048     'getInfo' 'ltpda_nuo';
```



```

0037      'gt' 'ltpda_nuo'; ...
0038      'isprop' 'ltpda_nuo'; ...
0039      'isvalid' 'ltpda_nuo'; ...
0040      'le' 'ltpda_nuo'; ...
0041      'lt' 'ltpda_nuo'; ...
0042      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_nuo/ltpda_nuo.m',
'ltpda_nuo', '1.16', '2008/09/03', '16:32:47', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_nuo',
'987912848e790406af21c62e8a4247d2', 'Constructor', 'ltpda_nuo'; ...
0043      'ne' 'ltpda_nuo'; ...
0044      'notify' 'ltpda_nuo'; ...
0045      'update_struct' 'ltpda_nuo'; ...
0046      'SETS' 'ltpda_uo'; ...
0047      'VEROUT' 'ltpda_uo'; ...
0048      'addlistener' 'ltpda_uo'; ...
0049      'char' 'ltpda_uo'; ...
0050      'copy' 'ltpda_uo'; ...
0051      'delete' 'ltpda_uo'; ...
0052      'display' 'ltpda_uo'; ...
0053      'eq' 'ltpda_uo'; ...
0054      'findobj' 'ltpda_uo'; ...
0055      'findprop' 'ltpda_uo'; ...
0056      'ge' 'ltpda_uo'; ...
0057      'getDefaultPlist' 'ltpda_uo'; ...
0058      'getInfo' 'ltpda_uo'; ...
0059      'gt' 'ltpda_uo'; ...
0060      'isprop' 'ltpda_uo'; ...
0061      'isvalid' 'ltpda_uo'; ...
0062      'le' 'ltpda_uo'; ...
0063      'lt' 'ltpda_uo'; ...
0064      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo/ltpda_uo.m', 'ltpda_uo',
'1.23', '2008/09/03', '16:33:56', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo',
'e789b8f5e29f8692362c151e51a8e5ae', 'Constructor', 'ltpda_uo'; ...
0065      'ne' 'ltpda_uo'; ...
0066      'notify' 'ltpda_uo'; ...
0067      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo/retrieve.m', 'retrieve',
'1.4', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo',
'0ab4be65bcc3c83add49260b916a5c6c', 'ltpda_uo'; ...
0068      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo/save.m',
'save', '1.5', '2008/09/07', '18:15:04', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo',
'efcce3bce7e27e2683c5f46db06c99ea', 'ltpda_uo'; ...
0069      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo/setName.m', 'setName',
'1.11', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo',
'eba9901979b3c883c43d4c67949e6658', 'ltpda_uo'; ...
0070      'string' 'ltpda_uo'; ...
0071      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo/submit.m',
'submit', '1.6', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo',
'1bb326194a19186cfd38d9aa9f018d5b', 'ltpda_uo'; ...
0072      'update_struct' 'ltpda_uo'; ...
0073      'SETS' 'ltpda_uoh'; ...
0074      'VEROUT' 'ltpda_uoh'; ...
0075      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh/addHistory.m',
'addHistory', '1.7', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh',
'a0feeaa29179f0a1bb40ba4c65a2a7055', 'ltpda_uoh'; ...
0076      'addlistener' 'ltpda_uoh'; ...
0077      'char' 'ltpda_uoh'; ...
0078      'copy' 'ltpda_uoh'; ...
0079      'delete' 'ltpda_uoh'; ...
0080      'display' 'ltpda_uoh'; ...
0081      'eq' 'ltpda_uoh'; ...
0082      'findobj' 'ltpda_uoh'; ...
0083      'findprop' 'ltpda_uoh'; ...
0084      'ge' 'ltpda_uoh'; ...
0085      'getDefaultPlist' 'ltpda_uoh'; ...
0086      'getInfo' 'ltpda_uoh'; ...
0087      'gt' 'ltpda_uoh'; ...
0088      'isprop' 'ltpda_uoh'; ...
0089      'isvalid' 'ltpda_uoh'; ...
0090      'le' 'ltpda_uoh'; ...
0091      'lt' 'ltpda_uoh'; ...
0092      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh/ltpda_uoh.m',
'ltpda_uoh', '1.14', '2008/09/03', '16:34:07', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh',
'c6d0b21f0af44e7815e87c9ce88a078f', 'Constructor', 'ltpda_uoh'; ...

```

```

0093      'ne' 'ltlda_uoh'; ...
0094      'notify' 'ltlda_uoh'; ...
0095      'retrieve' 'ltlda_uoh'; ...
0096      'save' 'ltlda_uoh'; ...
0097
0098      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@ltlda_uoh/setName.m', 'setName',
0099      '1.4', '2008/09/04', '15:29:30', 'ingo'
0100      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@ltlda_uoh',
0101      '35067a520f154855ae47198b5993f12d', 'ltlda_uoh'; ...
0102      'string', 'ltlda_uoh'; ...
0103      'submit', 'ltlda_uoh'; ...
0104      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@ltlda_uoh/type.m',
0105      'type', '1.5', '2008/09/04', '15:29:30', 'ingo'
0106      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@ltlda_uoh',
0107      '251f90e306de1c0aca4323cd3789dd03', 'ltlda_uoh'; ...
0108      'update_struct', 'ltlda_uoh'; ...
0109      'SETS', 'minfo'; ...
0110      'VEROUT', 'minfo'; ...
0111      'addlistener', 'minfo'; ...
0112      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo/char.m',
0113      'char', '1.4', '2008/09/04', '15:29:31', 'ingo'
0114      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo',
0115      'f87de39bfcf87bac667a07f96dfcd002', 'Output', 'minfo'; ...
0116      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo/copy.m',
0117      'copy', '1.6', '2008/09/03', '16:29:33', 'hewitson'
0118      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo',
0119      'd3e28273e2226345310788f67493dede', 'minfo'; ...
0120      'delete', 'minfo'; ...
0121      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo/display.m',
0122      'display', '1.4', '2008/09/04', '15:29:31', 'ingo'
0123      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo',
0124      'b101ad863ace7e111312e9d6e6c3d34b', 'Output', 'minfo'; ...
0125      'eq', 'Relational Operator', 'minfo'; ...
0126      'findobj', 'minfo'; ...
0127      'findprop', 'minfo'; ...
0128      'ge', 'minfo'; ...
0129      'getDefaultPlist', 'minfo'; ...
0130      'getInfo', 'minfo'; ...
0131      'gt', 'minfo'; ...
0132      'isprop', 'Helper', 'minfo'; ...
0133      'isvalid', 'minfo'; ...
0134      'le', 'minfo'; ...
0135      'lt', 'minfo'; ...
0136      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo/minfo.m',
0137      'minfo', '1.15', '2008/09/03', '16:34:49', 'hewitson'
0138      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo',
0139      '6699cdb4a8c770820b7609ef729b92a1', 'Constructor', 'minfo'; ...
0140      'ne', 'Relational Operator', 'minfo'; ...
0141      'notify', 'minfo'; ...
0142
0143      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo/setMversion.m',
0144      'setMversion', '1.3', '2008/09/04', '15:33:01', 'ingo'
0145      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo',
0146      'd4f45a76dba3e08e989b4db1232bebf5', 'Internal', 'minfo'; ...
0147
0148      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo/update_struct.m',
0149      'update_struct', '1.2', '2008/09/05', '14:16:11', 'hewitson'
0150      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@minfo',
0151      'bd616f809b3876703cc80d642fd83c61', 'minfo'; ...
0152      'SETS', 'unit'; ...
0153      'VEROUT', 'unit'; ...
0154      'addlistener', 'unit'; ...
0155      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@unit/char.m',
0156      'char', '1.1', '2008/09/05', '14:10:15', 'hewitson'
0157      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@unit',
0158      'e8056f4e00ce513fda35e18edb2229ac', 'unit'; ...
0159      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@unit/copy.m',
0160      'copy', '1.1', '2008/09/05', '14:10:15', 'hewitson'
0161      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@unit',
0162      'c0fd7c931b0d914a6f555f302ab65f93', 'unit'; ...
0163      'delete', 'unit'; ...
0164      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@unit/display.m',
0165      'display', '1.1', '2008/09/05', '14:10:15', 'hewitson'
0166      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@unit',
0167      'd91a459422a0f0da7f3414bef1df0455', 'Output', 'unit'; ...
0168      'eq', 'Relational Operator', 'unit'; ...
0169      'findobj', 'unit'; ...
0170      'findprop', 'unit'; ...
0171      'ge', 'unit'; ...
0172      'getDefaultPlist', 'unit'; ...
0173      'getInfo', 'unit'; ...
0174      'gt', 'unit'; ...
0175      'isprop', 'Helper', 'unit'; ...
0176      'isvalid', 'unit'; ...
0177      'le', 'unit'; ...
0178      'lt', 'unit'; ...
0179      '/Users/hewitson/working/matlab/ltlda_toolbox/ltlda/classes/@unit/mpower.m',

```



```
mpower', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'63b60fa6b2ef08f32653863b6b141e6d', 'unit', '...  
0144 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/mrdivide.m',  
'mrdivide', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit',  
'bd153d7a8b7f758f5b79fc5d4a8ff973', 'unit', '...  
0145 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/mtimes.m',  
'mtimes', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'b2a1241ab652786760d31c9bbb5f4441', 'unit', '...  
0146 'ne', 'Relational Operator', 'unit', '...  
0147 'notify', 'unit', '...  
0148 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/plus.m',  
'plus', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'd5f54788a271b7167fa4be2106391753', 'unit', '...  
0149 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/power.m',  
'power', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'8c0c9ec250c97e6bf9aca128090c581a', 'unit', '...  
0150 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/rdivide.m',  
'rdivide', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit',  
'3cf23d8ab411705c78425513ec216a2b', 'unit', '...  
0151 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/sqrt.m',  
'sqrt', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'691432674f72623651ef26365baf1139', 'unit', '...  
0152 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/string.m',  
'string', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit',  
'539d6aec863d4269269546095bd8cf5', 'Helper', 'unit', '...  
0153 'supportedPrefixes', 'unit', '...  
0154 'supportedUnits', 'unit', '...  
0155 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/times.m',  
'times', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'cf41d1550713239eb2c1913a2f03b3ec', 'unit', '...  
0156 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/unit.m',  
'unit', '1.3', '2008/09/08', '08:28:27', 'hewitson', 'unit', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit',  
'664b7801213098784a921cc410e6e90c', 'Constructor', 'unit', '...  
0157 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit/update_struct.m',  
'update_struct', '1.1', '2008/09/05', '14:10:15', 'hewitson', 'unit', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@unit',  
'bdbf05cd3b926c86f30bedc1eb566658', 'unit', '...  
0158 'SETS', 'ao', '...  
0159 'VEROUT', 'ao', '...  
0160 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/abs.m', 'abs',  
'1.33', '2008/09/06', '15:23:19', 'hewitson', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'c5d87c4688e878bf0440b89f006dadd7', 'Operator', 'ao', '...  
0161 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/acos.m', 'acos',  
'1.19', '2008/09/05', '11:05:28', 'ingo', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'ccb65d8f8693b92c4f2aed6585eba267', 'Trigonometry', 'ao', '...  
0162 'addHistory', 'Internal', 'ao', '...  
0163 'addlistener', 'ao', '...  
0164 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/ao.m', 'ao',  
'1.158', '2008/09/07', '18:11:06', 'hewitson', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'e04744b73298879a7fc1690f138de118', 'Constructor', 'ao', '...  
0165 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/ao2m.m', 'ao2m',  
'1.26', '2008/09/08', '09:00:39', 'hewitson', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'36bdb44e9a1cb71e97ded6d0f912963a', 'ao', '...  
0166 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/asin.m', 'asin',  
'1.16', '2008/09/05', '11:05:28', 'ingo', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'0491423510d626185d778a7713119af2', 'Trigonometry', 'ao', '...  
0167 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/atan.m', 'atan',  
'1.16', '2008/09/05', '11:05:28', 'ingo', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'51efc479d9ccb7aeaa55d4d6eec41ec', 'Trigonometry', 'ao', '...  
0168 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/atan2.m',  
'atan2', '1.8', '2008/09/05', '11:05:28', 'ingo', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'b21f2fe9acbee1f921899514dd5990b2', 'Trigonometry', 'ao', '...  
0169 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/attachm.m',  
'attachm', '1.14', '2008/09/05', '11:05:28', 'ingo', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'53fdd8c302fc9b0e24685b00f199f16a', 'Helper', 'ao', '...  
0170 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/attachmdl.m',  
'attachmdl', '1.14', '2008/09/05', '11:05:28', 'ingo', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'aea6e3b3279f60c044238c8f8ac2d702', 'Helper', 'ao', '...  
0171 'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao/cat.m', 'cat',  
'1.14', '2008/09/05', '11:05:28', 'ingo', 'ao', '...  
'Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@ao',  
'7ff6584409a6c04c32785a60db231111', 'Helper', 'ao', '...
```

```

0172      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/char.m', 'char',
'1.18', '2008/09/05', '11:05:28', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'35317947888f6617a8295454a8383ae7', 'Output', 'ao'; ...
0173      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/cohere.m',
'cohere', '1.10', '2008/09/05', '11:05:28', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'79d2380ae5d6c02b9fd12884a37b5b6', 'Signal Processing', 'ao'; ...
0174      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/complex.m',
'complex', '1.22', '2008/09/05', '14:13:27', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'75d61243cd4dcae666297b6a087c26f0', 'Operator', 'ao'; ...
0175      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/compute.m',
'compute', '1.11', '2008/09/05', '11:05:28', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'1309f2423e9390b09d74a0dabe4ea3a9', 'Signal Processing', 'ao'; ...
0176      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/conj.m', 'conj',
'1.23', '2008/09/05', '11:15:19', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'176a3edb9752b50995297062a3c1dce3', 'Operator', 'ao'; ...
0177      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/consolidate.m',
'consolidate', '1.14', '2008/09/05', '11:05:28', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'36bd550435230183eddb3d139098d4c3', 'Signal Processing', 'ao'; ...
0178      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/conv.m', 'conv',
'1.1', '2008/08/21', '09:01:03', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'24c33bd3013cf3620860521132fa6763', 'Signal Processing', 'ao'; ...
0179      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/copy.m', 'copy',
'1.14', '2008/09/03', '16:29:33', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'1acf17841f8d24499d7fd5d562682f29', 'ao'; ...
0180      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/cos.m', 'cos',
'1.15', '2008/09/05', '11:05:28', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'4a6531fe281970b0b58dd9187856ae32', 'Trigonometry', 'ao'; ...
0181      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/cpsd.m', 'cpsd',
'1.9', '2008/09/05', '11:05:28', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'b3b7c3c3db3feb9b87b7915f17ec1498', 'Signal Processing', 'ao'; ...
0182      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/ctranspose.m',
'ctranspose', '1.25', '2008/09/05', '11:15:19', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'fc13a93997ae759f73571edc72780e72', 'Operator', 'ao'; ...
0183      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/delay.m',
'delay', '1.12', '2008/09/05', '11:05:29', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'26ca00a58c33e53ec76d8ed19482915c', 'Signal Processing', 'ao'; ...
0184      'delete', 'ao'; ...
0185      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/demux.m',
'demux', '1.13', '2008/09/05', '11:05:29', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'fca70254a4cab4f59d0d4d9908ce4ec8', 'Helper', 'ao'; ...
0186      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/det.m', 'det',
'1.24', '2008/09/05', '11:15:19', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'c02f69e2ebf2ee21b69898e8604eab31', 'Operator', 'ao'; ...
0187      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/detrend.m',
'detrend', '1.10', '2008/09/05', '11:05:29', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'0ccac7ca7b070c78b1146b389156e00b', 'Signal Processing', 'ao'; ...
0188      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/dft.m', 'dft',
'1.13', '2008/09/05', '14:14:11', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'87ba3a52cda4c250531987fc92b1276a', 'Signal Processing', 'ao'; ...
0189      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/diag.m', 'diag',
'1.25', '2008/09/05', '11:15:19', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'0cc6b281c00f472d8c182d904b6066b2', 'Operator', 'ao'; ...
0190      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/diff.m', 'diff',
'1.10', '2008/09/05', '11:05:29', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'8fb474cba6622db425e3614dffbc8642', 'Signal Processing', 'ao'; ...
0191      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/display.m',
'display', '1.30', '2008/09/05', '11:05:29', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'05c09040c92dd7293cf0f957082255e5', 'Output', 'ao'; ...
0192      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/dopplercorr.m',
'dopplercorr', '1.6', '2008/09/05', '11:05:29', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'7e85708d6868ea30906a6eda952fefa5', 'Signal Processing', 'ao'; ...
0193      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/downsample.m',
'downsample', '1.15', '2008/09/05', '11:05:29', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'3d2e5c4d0def3765716f58c63c67f108', 'Signal Processing', 'ao'; ...
0194

```

```

'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/dropduplicates.m',
'dropduplicates', '1.10', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'61b491ca7afe7f59b79657728fbf1cd7', 'Signal Processing', 'ao'; ...
0195 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/dsmean.m',
'dsmean', '1.9', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'686077ce2c7f19c329b3fd4dce15aac2', 'Signal Processing', 'ao'; ...
0196 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/eig.m', 'eig',
'1.26', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'a5ad9e04a8fa78c898c83452f58f7cd0', 'Operator', 'ao'; ...
0197 'eq', 'Relational Operator', 'ao'; ...
0198 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/exp.m', 'exp',
'1.23', '2008/09/05', '14:11:47', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'54eb9d02b87028409f9dea65111ecd87', 'Operator', 'ao'; ...
0199 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/export.m',
'export', '1.15', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'0d44291b445fc75f93b3004f436ec9b5', 'Output', 'ao'; ...
0200 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/extractm.m',
'extractm', '1.7', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'e6fc533e700c646006bc90731ea43c64', 'Output', 'ao'; ...
0201 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/extractmdl.m',
'extractmdl', '1.7', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'b22a34b469cb57b32da557fca9143e50', 'Output', 'ao'; ...
0202 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/fft.m', 'fft',
'1.26', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'ba1f21ae5ae4e222c7c734729700123b', 'Signal Processing', 'ao'; ...
0203 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/filter.m',
'filter', '1.40', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'181df75c3edb26ef8645a462422f6b4d', 'Signal Processing', 'ao'; ...
0204 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/filtfilt.m',
'filtfilt', '1.27', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'70ae4207057f1166c28a1a95c3bd725b', 'Signal Processing', 'ao'; ...
0205 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/find.m', 'find',
'1.22', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'9fe8e652143c0f3578936ee0eda77e30', 'Helper', 'ao'; ...
0206 'findobj', 'ao'; ...
0207 'findprop', 'ao'; ...
0208 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/firwhiten.m',
'firwhiten', '1.10', '2008/09/08', '08:30:12', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'e5dc655e1b58e7658e19ab57f5dca986', 'Signal Processing', 'ao'; ...
0209 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/fixfs.m',
'fixfs', '1.11', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'52e4afd4a5df14d9f5356c2fb6c422b8', 'Signal Processing', 'ao'; ...
0210 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/fngen.m',
'fngen', '1.17', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'817718b5182828ac231cdded8516b483', 'Signal Processing', 'ao'; ...
0211 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/fs.m', 'fs',
'1.1', '2008/09/05', '13:55:17', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'241705f09420cdd5ef12d2788bb7220f', 'Internal', 'ao'; ...
0212 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/gapfilling.m',
'gapfilling', '1.5', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'2fc05982a0d467488fe07ea4f43c3642', 'Signal Processing', 'ao'; ...
0213 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/ge.m', 'ge',
'1.14', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'3959050600d2c0684c031503812102bd', 'Relational Operator', 'ao'; ...
0214 'getDefaultPlist', 'ao'; ...
0215 'getInfo', 'ao'; ...
0216 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/gt.m', 'gt',
'1.14', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'f0add9902445f52ba8b7838abbc919e4', 'Relational Operator', 'ao'; ...
0217 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/hist.m', 'hist',
'1.23', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'8cddfa8cc3e73e7f38433e0bc75d6a4f', 'Signal Processing', 'ao'; ...
0218 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/iff.m', 'iff',
'1.11', '2008/09/08', '08:30:56', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'a1a752b408f636be27812e1578e0658c', 'Signal Processing', 'ao'; ...
0219 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/imag.m', 'imag',

```



```

1.12', '2008/09/05', '11:15:19', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'18db7133aed7a91aab34c0752b9a05f5', 'Operator', 'ao'; ...
0220
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/index.m',
'index', '1.17', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'b1eeea181d83a23869daaf2aed3e18660', 'Helper', 'ao'; ...
0221
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/interp.m',
'interp', '1.24', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'c18a0677ceb1e58e1c02d2b04ebfd269', 'Signal Processing', 'ao'; ...
0222
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/interpmissing.m',
'interpmissing', '1.12', '2008/09/08', '08:31:18', 'hewitson'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'44f2a978a215f460f0ddda05e83ab9ff', 'Signal Processing', 'ao'; ...
0223
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/inv.m', 'inv',
'1.22', '2008/09/05', '11:15:19', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'15baf0f9c677dba5646c760a7d4befb4', 'Operator', 'ao'; ...
0224
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/iplot.m',
'iplot', '1.48', '2008/09/05', '14:14:33', 'hewitson'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'bfbcb281f3e05dca2296f95a82f94fac', 'Output', 'ao'; ...
0225
'isprop', 'Helper', 'ao'; ...
0226
'isvalid', 'Helper', 'ao'; ...
0227
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/join.m', 'join',
'1.24', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'6d6de67056b270ea301f90cf9df12a70', 'Helper', 'ao'; ...
0228
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/lcohere.m',
'lcohere', '1.7', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'b1a886aafe0837d07b1dc57a5f40ebc0', 'Signal Processing', 'ao'; ...
0229
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/lcpsd.m',
'lcpsd', '1.7', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'08426f3a172c2486cde8ff55e152dda8', 'Signal Processing', 'ao'; ...
0230
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/le.m', 'le',
'1.15', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'5111fefb72280ae8016532642cd4420a', 'Relational Operator', 'ao'; ...
0231
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/len.m', 'len',
'1.19', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'5483395b2033bc9fb94abf80aaa67e49', 'Helper', 'ao'; ...
0232
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/lincom.m',
'lincom', '1.9', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'a9fa4582e9fd5be15dc280162fc0a2fe', 'Signal Processing', 'ao'; ...
0233
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/linedetect.m',
'linedetect', '1.3', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'e08edf8ad4cfc3e2480ac06e3002a535', 'Signal Processing', 'ao'; ...
0234
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/ln.m', 'ln',
'1.17', '2008/09/05', '14:11:47', 'hewitson'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'3fd5cfb97f32ad1a523f0779a3a8f00c', 'Operator', 'ao'; ...
0235
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/log.m', 'log',
'1.22', '2008/09/05', '14:11:47', 'hewitson'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'1e98dfd8cf3ce371095ed77e1343c73e', 'Operator', 'ao'; ...
0236
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/log10.m',
'log10', '1.19', '2008/09/05', '14:11:47', 'hewitson'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'07dc5b9c66c6ec3142b28fe34171352c', 'Operator', 'ao'; ...
0237
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/lpsd.m', 'lpsd',
'1.13', '2008/09/05', '14:14:58', 'hewitson'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'1747c07da106ca119cc065273d1951e0', 'Signal Processing', 'ao'; ...
0238
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/lt.m', 'lt',
'1.15', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'3714c790025cbe3911ff2210ed18d4c7', 'Relational Operator', 'ao'; ...
0239
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/ltfe.m', 'ltfe',
'1.6', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'768b96e211dfcb9e59da18443ff4b20b', 'Signal Processing', 'ao'; ...
0240
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/max.m', 'max',
'1.13', '2008/09/08', '08:31:56', 'hewitson'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'63582bd38332f14cfe2afd393bfc64b', 'Operator', 'ao'; ...
0241
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao/md5.m', 'md5',
'1.12', '2008/09/05', '11:05:29', 'ingo'
/Users/hewitson/working/matlab/ltplib/toolbox/ltplib/classes/@ao',
'6236c632c00016cbb8246f2792adcf58', 'Helper', 'ao'; ...

```

```

0242
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mdc1_ifo2acc_fd.m',
'mdc1_ifo2acc_fd', '1.4', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'adec6d07fb440fe3bf7283ae0aee38b5', 'MDC01', 'ao'; ...
0243
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mdc1_ifo2acc_inloop.m',
'mdc1_ifo2acc_inloop', '1.10', '2008/09/05', '14:15:14', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'6cadcd771b346ac13246250a84530509', 'MDC01', 'ao'; ...
0244
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mdc1_ifo2control.m',
'mdc1_ifo2control', '1.8', '2008/09/05', '14:15:14', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'ad7b7423e4b91a840ea00994e40536bd', 'MDC01', 'ao'; ...
0245
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mdc1_input_noises.m',
'mdc1_input_noises', '1.3', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'7a0c4c6cfde0d83f9fa406b7514202b3', 'ao'; ...
0246
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mdc1_x2acc.m',
'mdc1_x2acc', '1.7', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'ae82ac7df3b451856483bb7e44166473', 'MDC01', 'ao'; ...
0247
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mean.m', 'mean',
'1.24', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'044849bbec16657e888055499a2b943d', 'Operator', 'ao'; ...
0248
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/median.m',
'median', '1.19', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'f0420db33fd5ba08bad2df9d56deb4f3', 'Operator', 'ao'; ...
0249
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/min.m', 'min',
'1.14', '2008/09/08', '08:31:56', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'c66a9c6e8af5e48962326d677268a1cf', 'Operator', 'ao'; ...
0250
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/minus.m',
'minus', '1.24', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'537f1f4c4cc0e1ae63d634bb773eedf3', 'Arithmetic Operator', 'ao'; ...
0251
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mode.m', 'mode',
'1.9', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'7dd184c466f0cb359c18c0f580d76426', 'Operator', 'ao'; ...
0252
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mpower.m',
'mpower', '1.24', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'0358a8f5f6b308774d5d774ecfc43b53', 'Arithmetic Operator', 'ao'; ...
0253
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mrdivide.m',
'mrdivide', '1.24', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'd07dd12540c2af1f79aefca01e098898', 'Arithmetic Operator', 'ao'; ...
0254
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/mtimes.m',
'mtimes', '1.28', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'a95c5d5e9b87cda06993a85229e423fb', 'Arithmetic Operator', 'ao'; ...
0255
'', 'ne', 'Relational Operator', 'ao'; ...
0256
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/norm.m', 'norm',
'1.22', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'd4ce1e89039aac6cf7fbd2c0739aa14a', 'Operator', 'ao'; ...
0257
'', 'notify', 'ao'; ...
0258
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/phase.m',
'phase', '1.12', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'830658b17de6cc0c631cefa4861d4b13', 'Operator', 'ao'; ...
0259
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/plot.m', 'plot',
'1.36', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'26ac925364200c9e76bb4c61ef452473', 'Output', 'ao'; ...
0260
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/plus.m', 'plus',
'1.37', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'b7572d9e234e5b66d110952cd6f81947', 'Arithmetic Operator', 'ao'; ...
0261
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/polyfit.m',
'polyfit', '1.18', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'a9bb7e92be96a14ec3685c1eea6341bb', 'Signal Processing', 'ao'; ...
0262
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/power.m',
'power', '1.22', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'2a29cd32dc0fb9177db419463dc9faf1', 'Arithmetic Operator', 'ao'; ...
0263
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/psd.m', 'psd',
'1.6', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'bf4e0196aa046c6a0181bbebf6e174b9', 'Signal Processing', 'ao'; ...

```

```

0264      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/pwelch.m',
      'pwelch', '1.14', '2008/08/07', '13:00:39', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '2cf6bcc3532b202363266e260e946d94', 'Signal Processing', 'ao'; ...
0265      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/rdivide.m',
      'rdivide', '1.24', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '61066fe4b8b7727053bc5708f127110b', 'Arithmetic Operator', 'ao'; ...
0266      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/real.m', 'real',
      '1.11', '2008/09/05', '11:15:19', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'cbe903f1ff99400a54d49b1da44bbabf', 'Operator', 'ao'; ...
0267      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/resample.m',
      'resample', '1.30', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '9ebad40721ef5221d3c1ff8a64008290', 'Signal Processing', 'ao'; ...
0268      'retrieve', '', 'Signal Processing', 'ao'; ...
0269      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/rms.m', 'rms',
      '1.9', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '80653396e45574b38aef37737194bcbe', 'Signal Processing', 'ao'; ...
0270      'save', '', 'Output', 'ao'; ...
0271      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/search.m',
      'search', '1.4', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'c7d8d4f6bb680a59e9a739f7927fc23d', 'Helper', 'ao'; ...
0272      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/select.m',
      'select', '1.26', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '9b72fddbcf9bc855170aaecb5ab48fda', 'Signal Processing', 'ao'; ...
0273      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setDescription.m',
      'setDescription', '1.7', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '9bec8e1a05af31b6a9cb64cd56071756', 'Helper', 'ao'; ...
0274      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setFs.m',
      'setFs', '1.6', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '4e005738743421708eb5b3bc68657419', 'Helper', 'ao'; ...
0275      'setName', '', 'Helper', 'ao'; ...
0276      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setT0.m',
      'setT0', '1.6', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'f428c65ea733f2420875374b69c3ae', 'Helper', 'ao'; ...
0277      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setX.m', 'setX',
      '1.6', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'fae8e191438f1f1c4bbaf87412af44d8', 'Helper', 'ao'; ...
0278      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setXY.m',
      'setXY', '1.6', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'd303c12a218f7d7741bf4754cd2c5ef1', 'Helper', 'ao'; ...
0279      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setXunits.m',
      'setXunits', '1.9', '2008/09/05', '14:15:33', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'ea3b0595467a43a00e0a8f2ba2366fe5', 'Helper', 'ao'; ...
0280      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setY.m', 'setY',
      '1.6', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '55d331163cdd100c67f5111b07eaf7d', 'Helper', 'ao'; ...
0281      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setYunits.m',
      'setYunits', '1.9', '2008/09/05', '14:15:33', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'f8cea923b169f0d9a95e62bf971f7d7a', 'Helper', 'ao'; ...
0282      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/setZ.m', 'setZ',
      '1.6', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'd417ccf5478d1abdf1d88447689054d1', 'Helper', 'ao'; ...
0283      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/sign.m', 'sign',
      '1.7', '2008/09/05', '11:15:19', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'e78a16556384ccd83012ad43909fc060', 'Operator', 'ao'; ...
0284      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/sin.m', 'sin',
      '1.23', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'b641032e90a0f9414dfd197d9df5daff', 'Trigonometry', 'ao'; ...
0285      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/smooth.m',
      'smoother', '1.12', '2008/09/05', '11:05:29', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      'b4b2142691fb06dbb61f6ed6dcfbada4', 'Signal Processing', 'ao'; ...
0286      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/sort.m', 'sort',
      '1.10', '2008/09/05', '11:15:19', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
      '2a8c824b4e2c8b91c87bad828304c529', 'Operator', 'ao'; ...
0287      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/spectrogram.m',
      'spectrogram', '1.10', '2008/09/05', '14:15:40', 'hewitson',

```



```

'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'0e1f6e6495cb2b92981b3dd96e043d4a', 'Signal Processing', 'ao'; ...
0288
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/spikecleaning.m',
'spikecleaning', '1.6', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'ec75a91e9943b0377bf158edd86bc55e', 'Signal Processing', 'ao'; ...
0289
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/split.m',
'split', '1.56', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'3162b337e230e298be561b43b37a9e90', 'Signal Processing', 'ao'; ...
0290
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/sqrt.m', 'sqrt',
'1.24', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'598f35d02b6e94dab5e5247a094a1652', 'Operator', 'ao'; ...
0291
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/std.m', 'std',
'1.18', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'34e78bf5ffbf6b697f04e895a8b82aa', 'Operator', 'ao'; ...
0292
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/string.m',
'string', '1.10', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'607d22a1df5fdfec9dce50b316f67862', 'Helper', 'ao'; ...
0293
'', 'submit', 'Internal', 'ao'; ...
0294
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/sum.m', 'sum',
'1.15', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'2a28f94f583d8abbddaa8dff1c59ae3e', 'Operator', 'ao'; ...
0295
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/svd.m', 'svd',
'1.19', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'9b3bf604c2bded2b2565c5d6a884879b', 'Operator', 'ao'; ...
0296
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/t0.m', 't0',
'1.1', '2008/09/05', '13:55:17', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'882ef8b7971a1c9b974022de7f591e0d', 'Internal', 'ao'; ...
0297
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/tan.m', 'tan',
'1.15', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'3a1600ab13151de811d7a79efab40ed6', 'Trigonometry', 'ao'; ...
0298
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/tfe.m', 'tfe',
'1.6', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'855b928f17318466bc81a7370f129c16', 'Signal Processing', 'ao'; ...
0299
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/timedomainfit.m',
'timedomainfit', '1.6', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'7bc0468b3c203dbbee12dd988c7a3f52c', 'Signal Processing', 'ao'; ...
0300
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/times.m',
'times', '1.25', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'683fbcd5977439ee8e5a0e9d0772562a', 'Arithmetic Operator', 'ao'; ...
0301
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/timeshift.m',
'timeshift', '1.11', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'74bd1b96887793fa3efc11e4be8448b4', 'Helper', 'ao'; ...
0302
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/transpose.m',
'transpose', '1.22', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'd60e14b41591653a687a49bd5de5450f', 'Operator', 'ao'; ...
0303
'', 'type', 'Output', 'ao'; ...
0304
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/uminus.m',
'uminus', '1.7', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'e27c504962703468a6aba94e7e17a196', 'Operator', 'ao'; ...
0305
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/update_struct.m',
'update_struct', '1.2', '2008/08/01', '13:19:42', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'fb71407b37cba221d8df807d043f453b', 'ao'; ...
0306
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/upsample.m',
'upsample', '1.12', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'b62c0af7eb2199ae19b23e0ec994bdb1', 'Signal Processing', 'ao'; ...
0307
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/validate.m',
'validate', '1.6', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'c10483cca9e9c47e2162c89577e27696', 'Helper', 'ao'; ...
0308
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/var.m', 'var',
'1.18', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'3ebbad9a86cc1ab17bc3e5445e59f1b0', 'Operator', 'ao'; ...
0309
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/x.m', 'x',
'1.1', '2008/09/05', '13:55:17', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',

```

```

'1a187ce2f371b34a059750dc2f1183f2', 'Internal', 'ao'; ...
0310 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/xcorr.m',
'xcorr', '1.5', '2008/09/05', '14:15:40', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'75a75a938379af0c6f5ee9ecc3e58812', 'Signal Processing', 'ao'; ...
0311 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/xunits.m',
'xunits', '1.1', '2008/09/05', '13:55:17', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'ee73f32fd2a091f32f27cfbb53b29e85', 'Internal', 'ao'; ...
0312 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/y.m', 'y',
'1.1', '2008/09/05', '13:55:17', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'6882e1deba053654e5308952faf5718b', 'Internal', 'ao'; ...
0313 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/yunits.m',
'yunits', '1.1', '2008/09/05', '13:55:17', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'89127477dc514214b4f3e98d4ecf1c4', 'Internal', 'ao'; ...
0314 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/zeropad.m',
'zeropad', '1.11', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'd1102d4ec171ce8d1bf019842d90b229', 'Operator', 'ao'; ...
0315 'SETS', 'param'; ...
0316 'VEROUT', 'param'; ...
0317 'addlistener', 'param'; ...
0318 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/char.m',
'char', '1.17', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'cd73ee50ee8ba0fe2366969c731db1fc', 'Output', 'param'; ...
0319 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'f4c9bee86b91eab7002b04760ccb8eb9', 'param'; ...
0320 'delete', 'param'; ...
0321 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/display.m',
'display', '1.36', '2008/09/07', '10:54:11', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'dbaa5240241ea70677d271d66687a5f9', 'Output', 'param'; ...
0322 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/display2.m',
'display2', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'9599abc27cb7a80a13b4b0892ad165a2', 'Output', 'param'; ...
0323 'eq', 'Relational Operator', 'param'; ...
0324 'findobj', 'param'; ...
0325 'findprop', 'param'; ...
0326 'ge', 'param'; ...
0327 'getDefaultPlist', 'param'; ...
0328 'getInfo', 'param'; ...
0329 'gt', 'param'; ...
0330 'isprop', 'Helper', 'param'; ...
0331 'isvalid', 'param'; ...
0332 'le', 'param'; ...
0333 'lt', 'param'; ...
0334 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/mux.m',
'mux', '1.10', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'b8682d098b96eea716ad6c1f884f7e6e', 'Helper', 'param'; ...
0335 'ne', 'Relational Operator', 'param'; ...
0336 'notify', 'param'; ...
0337 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/param.m',
'param', '1.44', '2008/09/03', '16:35:22', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'b8290a811b7c05cf2cb8221fa569e38a', 'Constructor', 'param'; ...
0338 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/setKey.m',
'setKey', '1.6', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'bb762c9b18aa327bfff2e15b4a3d6c80b', 'Internal', 'param'; ...
0339 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/setKeyVal.m',
'setKeyVal', '1.6', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'c21530b89df582b9a0f8319b13d64dde', 'Internal', 'param'; ...
0340 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/setVal.m',
'setVal', '1.6', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'e3bb1f9eef33abf58a2ed3ce2839e6e5', 'Internal', 'param'; ...
0341 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/string.m',
'string', '1.7', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'9bc0baf7f57f9c171fd125ee9c865f39', 'Output', 'param'; ...
0342 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:49', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@param',
'2cffcaf9b7b9a583880ae4fa8013c826', 'param'; ...
0343 'SETS', 'plist'; ...
0344 'VEROUT', 'plist'; ...
0345 'addlistener', 'plist'; ...

```



```

0346      '/Users/hewitson/working/matlab/ltppda/classes/@plist/append.m',
'append', '1.17', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'c4c85fdb6fe4afa04281f65683ec7bc', 'Helper', 'plist'; ...
0347      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/char.m',
'char', '1.21', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'100eccbc4ad297a1ac6c3775eeeed636', 'Output', 'plist'; ...
0348      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/combine.m',
'combine', '1.17', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'e2bb875d521c6161fbce86e7d45398d3', 'Helper', 'plist'; ...
0349      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'0a8bb40007a76e20aba4040777c209a3', 'plist'; ...
0350      'delete', 'plist'; ...
0351      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/display.m',
'display', '1.11', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'09b5fd66a27fd30e7e327f7877744867', 'Output', 'plist'; ...
0352      'eq', 'Relational Operator', 'plist'; ...
0353      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/find.m',
'find', '1.13', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'0cda2c3d6ead7302464bb1c13edbaaa0', 'Helper', 'plist'; ...
0354      'findobj', 'plist'; ...
0355      'findprop', 'plist'; ...
0356      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/fromFile.m',
'fromFile', '1.2', '2008/09/08', '09:40:46', 'hewitson'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'87e0325e6fbc4a61a7d02fd7ae10433e', 'plist'; ...
0357      'ge', 'plist'; ...
0358      'getDefaultPlist', 'plist'; ...
0359      'getInfo', 'plist'; ...
0360      'gt', 'plist'; ...
0361      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/isparam.m',
'isparam', '1.9', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'87e6be4c27f6e8d8fa4466eda86e0feb', 'Helper', 'plist'; ...
0362      'isprop', 'Helper', 'plist'; ...
0363      'isvalid', 'plist'; ...
0364      'le', 'plist'; ...
0365      'lt', 'plist'; ...
0366      'ne', 'Relational Operator', 'plist'; ...
0367      'notify', 'plist'; ...
0368      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/nparams.m',
'nparams', '1.10', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'6cce5f90aee6eec8171c47b4697ff009', 'Helper', 'plist'; ...
0369      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/plist.m',
'plist', '1.51', '2008/09/07', '18:15:26', 'hewitson'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'c031a1b8ef50ec0366dea284029ba9f8', 'Constructor', 'plist'; ...
0370      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/pset.m',
'pset', '1.11', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'0670334d0952be20a1095a489cf256ad', 'Helper', 'plist'; ...
0371      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/remove.m',
'remove', '1.13', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'a1631d41eeb2417be59d36813aac8e20', 'Helper', 'plist'; ...
0372      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/resp.m',
'resp', '1.15', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'd107125003c05e303b6bd2ac61e18adb', 'Signal Processing', 'plist'; ...
0373      'retrieve', 'plist'; ...
0374      'save', 'Output', 'plist'; ...
0375      'setName', 'Helper', 'plist'; ...
0376      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/string.m',
'string', '1.21', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'ddc543d069a2569cc0efd0836bce9ab1', 'Helper', 'plist'; ...
0377      'submit', 'Internal', 'plist'; ...
0378      '/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:49', 'ingo'
'/Users/hewitson/working/matlab/ltppda_toolbox/ltppda/classes/@plist',
'd0a17bfcf7fa1f0f160996ff795c7113', 'plist'; ...
0379      'SETS', 'ltppda_data'; ...
0380      'VEROUT', 'ltppda_data'; ...
0381      'addlistener', 'ltppda_data'; ...
0382      'char', 'ltppda_data'; ...
0383      'copy', 'ltppda_data'; ...
0384      'delete', 'ltppda_data'; ...
0385      'display', 'ltppda_data'; ...

```

```

0386      'eq' 'ltpda_data'; ...
0387      'findobj' 'ltpda_data'; ...
0388      'findprop' 'ltpda_data'; ...
0389      'ge' 'ltpda_data'; ...
0390      'getDefaultPlist' 'ltpda_data'; ...
0391      'getInfo' 'ltpda_data'; ...
0392      'gt' 'ltpda_data'; ...
0393      'isprop' 'ltpda_data'; ...
0394      'isvalid' 'ltpda_data'; ...
0395      'le' 'ltpda_data'; ...
0396      'lt' 'ltpda_data'; ...
0397
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_data/ltpda_data.m',
'ltpda_data', '1.10', '2008/09/03', '16:32:48', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_data',
'b40e894705074be6a3f6425ba20c2da2', 'Constructor', 'ltpda_data'; ...
0398      'ne' 'ltpda_data'; ...
0399      'notify' 'ltpda_data'; ...
0400      'update_struct' 'ltpda_data'; ...
0401      'SETS' 'data2D'; ...
0402      'VEROUT' 'data2D'; ...
0403      'addlistener' 'data2D'; ...
0404
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/applymethod.m',
'applymethod', '1.4', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'd2bc399a8d770eecd924915da71038e', 'data2D'; ...
0405
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/applyoperator.m',
'applyoperator', '1.12', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'0f0c9d54d4ba5eb65361863d8dfddbfb', 'data2D'; ...
0406
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/char.m',
'char', '1.4', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'd5fb3ac01d22e1443557550cb934a853', 'data2D'; ...
0407      'copy' 'data2D'; ...
0408
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/data2D.m',
'data2D', '1.23', '2008/09/07', '18:11:58', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'4482a229ba4e0b7328ee9943aa4736d8', 'Constructor', 'data2D'; ...
0409      'delete' 'data2D'; ...
0410      'display' 'data2D'; ...
0411      'eq' 'data2D'; ...
0412      'findobj' 'data2D'; ...
0413      'findprop' 'data2D'; ...
0414      'ge' 'data2D'; ...
0415      'getDefaultPlist' 'data2D'; ...
0416      'getInfo' 'data2D'; ...
0417
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/getX.m',
'getX', '1.4', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'722972601960df07fd5504c93360cf31', 'data2D'; ...
0418
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/getY.m',
'getY', '1.3', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'571925958e780afab54839ec0810b28a', 'data2D'; ...
0419      'gt' 'data2D'; ...
0420      'isprop' 'data2D'; ...
0421      'isvalid' 'data2D'; ...
0422      'le' 'data2D'; ...
0423      'lt' 'data2D'; ...
0424      'ne' 'data2D'; ...
0425      'notify' 'data2D'; ...
0426
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/setX.m',
'setX', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'ce60ee82d9f7f2fc03c6a27b42bc04f8', 'data2D'; ...
0427
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/setXY.m',
'setXY', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'42fcff88d2a7a9750d4261902d0659cb', 'data2D'; ...
0428
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/setXunits.m', 'setXunits',
'1.6', '2008/09/05', '14:16:30', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'b7ebbdde4995d774529a57850d7a98ab', 'data2D'; ...
0429
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/setY.m',
'setY', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'87c636db7aecc6d7bf21e9d2d87a544b', 'data2D'; ...
0430
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D/setYunits.m', 'setYunits',
'1.6', '2008/09/05', '14:16:30', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D',
'739f3c54e60bbac3a90aed6a6ce041a3', 'data2D'; ...

```

```

0431      'update_struct', 'Internal', 'cdata'; ...
0432      'SETS', 'Internal', 'cdata'; ...
0433      'VEROUT', 'Internal', 'cdata'; ...
0434      'addlistener', 'Internal', 'cdata'; ...
0435      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/applymethod.m',
      'applymethod', '1.4', '2008/09/04', '15:29:30', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      'ce46ebd1095ea40205f1aaed5a938101', 'Internal', 'cdata'; ...
0436      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/applyoperator.m',
      'applyoperator', '1.8', '2008/09/04', '15:29:30', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '369ce53ec5398868fbd772289002f7c8', 'Internal', 'cdata'; ...
0437      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/cdata.m',
      'cdata', '1.47', '2008/09/07', '18:11:30', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '45e948cb0d264eac3a039fca6df17155', 'Constructor', 'cdata'; ...
0438      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/char.m',
      'char', '1.9', '2008/09/04', '15:29:30', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      'b02eac4e95998ef640817bbb1961b334', 'Internal', 'cdata'; ...
0439      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/copy.m',
      'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '46384efa4c8efe3847a61d41c13af836', 'Internal', 'cdata'; ...
0440      'delete', 'Internal', 'cdata'; ...
0441      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/display.m',
      'display', '1.14', '2008/09/05', '14:17:51', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '2f0491cb162980b81aef86e46783d6e2', 'Output', 'cdata'; ...
0442      'eq', 'Relational Operator', 'cdata'; ...
0443      'findobj', 'Internal', 'cdata'; ...
0444      'findprop', 'Internal', 'cdata'; ...
0445      'ge', 'Internal', 'cdata'; ...
0446      'getDefaultPlist', 'Internal', 'cdata'; ...
0447      'getInfo', 'Internal', 'cdata'; ...
0448      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/getY.m',
      'getY', '1.2', '2008/09/04', '15:29:30', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '8bad55167c61564f238d9ebfa0039109', 'Internal', 'cdata'; ...
0449      'gt', 'Internal', 'cdata'; ...
0450      'isprop', 'Helper', 'cdata'; ...
0451      'isvalid', 'Internal', 'cdata'; ...
0452      'le', 'Internal', 'cdata'; ...
0453      'lt', 'Internal', 'cdata'; ...
0454      'ne', 'Relational Operator', 'cdata'; ...
0455      'notify', 'Internal', 'cdata'; ...
0456      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/setY.m',
      'setY', '1.2', '2008/09/04', '15:29:30', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '12cf7e0b4202452403322289189afac8', 'Internal', 'cdata'; ...
0457      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/setYunits.m',
      'setYunits', '1.4', '2008/09/05', '14:16:30', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '4a5e56a9880abc21fb58f5982d4fbeb', 'Internal', 'cdata'; ...
0458      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata/update_struct.m',
      'update_struct', '1.2', '2008/09/05', '14:16:11', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata',
      '35e36bc6676b683c83056e758bab9bb3', 'Internal', 'cdata'; ...
0459      'SETS', 'Internal', 'tsdata'; ...
0460      'VEROUT', 'Internal', 'tsdata'; ...
0461      'addlistener', 'Internal', 'tsdata'; ...
0462      'applymethod', 'Internal', 'tsdata'; ...
0463      'applyoperator', 'Internal', 'tsdata'; ...
0464      'char', 'Output', 'tsdata'; ...
0465      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/collapseX.m', 'collapseX',
      '1.2', '2008/09/04', '15:29:32', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
      'bd5cc32e65f619a00ca8e393becf2d37', 'Internal', 'tsdata'; ...
0466      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/copy.m',
      'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
      '31b112d9fd6a692160a26c4a9ef9c4da', 'Internal', 'tsdata'; ...
0467      'delete', 'Internal', 'tsdata'; ...
0468      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/display.m',
      'display', '1.18', '2008/09/04', '15:29:32', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
      '00e4b0ead11a23be53ebb4aa4ba39405', 'Output', 'tsdata'; ...
0469      'eq', 'Relational Operator', 'tsdata'; ...
0470      'findobj', 'Internal', 'tsdata'; ...
0471      'findprop', 'Internal', 'tsdata'; ...
0472      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/fitfs.m',
      'fitfs', '1.7', '2008/09/04', '14:14:02', 'hewitson',

```



```

'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'815eb5d4319d767e70d2b8f1625d35a6', 'tsdata'; ...
0473 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/fixNsecs.m',
'fixNsecs', '1.2', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'ea519cdb6a059be6ee4feb9899e12e86', 'Internal', 'tsdata'; ...
0474 'ge', 'tsdata'; ...
0475 'getDefaultPlist', 'tsdata'; ...
0476 'getInfo', 'tsdata'; ...
0477 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/getX.m',
'getX', '1.4', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'eade61374e418f15d5b701932f2157b5', 'Internal', 'tsdata'; ...
0478 'getY', 'Internal', 'tsdata'; ...
0479 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/growT.m',
'growT', '1.3', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'82e747a66d05248592a6e321658e190c', 'Internal', 'tsdata'; ...
0480 'gt', 'tsdata'; ...
0481 'isprop', 'Helper', 'tsdata'; ...
0482 'isvalid', 'tsdata'; ...
0483 'le', 'tsdata'; ...
0484 'lt', 'tsdata'; ...
0485 'ne', 'Relational Operator', 'tsdata'; ...
0486 'notify', 'tsdata'; ...
0487 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/setFs.m',
'setFs', '1.2', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'c1f2f598b6a43dbf732f5da01c772d1a', 'Internal', 'tsdata'; ...
0488 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/setNsecs.m',
'setNsecs', '1.2', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'598d05a08dbd9a3035fc9d5acc74991', 'Internal', 'tsdata'; ...
0489 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/setT0.m',
'setT0', '1.2', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'cf3bc6478ea3b77f39a52df98c5a4e75', 'Internal', 'tsdata'; ...
0490 'setX', 'Internal', 'tsdata'; ...
0491 'setXY', 'Internal', 'tsdata'; ...
0492 'setXunits', 'Internal', 'tsdata'; ...
0493 'setY', 'Internal', 'tsdata'; ...
0494 'setYunits', 'Internal', 'tsdata'; ...
0495 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/tsdata.m',
'tsdata', '1.57', '2008/09/03', '16:39:37', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'3d33426abc70c5081d1ae29ebf26b902', 'Constructor', 'tsdata'; ...
0496
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata/update_struct.m',
'update_struct', '1.3', '2008/09/05', '14:16:11', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@tsdata',
'85d6da9b1caafa3d1f51aacc1d9507ed', 'tsdata'; ...
0497 'SETS', 'fsdata'; ...
0498 'VEROUT', 'fsdata'; ...
0499 'addListener', 'fsdata'; ...
0500 'applymethod', 'Internal', 'fsdata'; ...
0501 'applyoperator', 'Internal', 'fsdata'; ...
0502 'char', 'Output', 'fsdata'; ...
0503 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'ae31f9f02aef5e1f73cb8e6b21854fa5', 'fsdata'; ...
0504 'delete', 'fsdata'; ...
0505 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/display.m',
'display', '1.14', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'6c67f0ba6f18a393374f2adacc2e5953', 'Output', 'fsdata'; ...
0506 'eq', 'Relational Operator', 'fsdata'; ...
0507 'findobj', 'fsdata'; ...
0508 'findprop', 'fsdata'; ...
0509 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/fsdata.m',
'fsdata', '1.39', '2008/09/03', '16:32:18', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'683a05b19e55e8d4c5cf79b860ed4b7d', 'Constructor', 'fsdata'; ...
0510 'ge', 'fsdata'; ...
0511 'getDefaultPlist', 'fsdata'; ...
0512 'getInfo', 'fsdata'; ...
0513 'getX', 'Internal', 'fsdata'; ...
0514 'getY', 'Internal', 'fsdata'; ...
0515 'gt', 'fsdata'; ...
0516 'isprop', 'Helper', 'fsdata'; ...
0517 'isvalid', 'fsdata'; ...
0518 'le', 'fsdata'; ...
0519 'lt', 'fsdata'; ...
0520 'ne', 'Relational Operator', 'fsdata'; ...
0521 'notify', 'fsdata'; ...
0522 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/setEnbw.m',

```

```

'setEnbw', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'2333c4c448dd2143830d3430264f4fd9', 'Internal', 'fsdata'; ...
0523 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/setFs.m',
'setFs', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'a549c2efff70181d2406f9da8380a862', 'Internal', 'fsdata'; ...
0524 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/setNavs.m',
'setNavs', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'4e2a8d91ac1fcf5abe0e56a21610e5ce', 'Internal', 'fsdata'; ...
0525 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/setT0.m',
'setT0', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'f7099c9c6d027bca95e4ae138b820ffd', 'Internal', 'fsdata'; ...
0526 'setX', 'Internal', 'fsdata'; ...
0527 'setXY', 'Internal', 'fsdata'; ...
0528 'setXunits', 'Internal', 'fsdata'; ...
0529 'setY', 'Internal', 'fsdata'; ...
0530 'setYunits', 'Internal', 'fsdata'; ...
0531 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata/update_struct.m',
'update_struct', '1.3', '2008/09/05', '14:16:11', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@fsdata',
'2280421453225b6a89f9375b05c488db', 'fsdata'; ...
0532 'SETS', 'xydata'; ...
0533 'VEROUT', 'xydata'; ...
0534 'addlistener', 'xydata'; ...
0535 'applymethod', 'Internal', 'xydata'; ...
0536 'applyoperator', 'Internal', 'xydata'; ...
0537 'char', 'Output', 'xydata'; ...
0538 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata',
'7448204acdf301fe9ef1eeaf67f77cae', 'xydata'; ...
0539 'delete', 'xydata'; ...
0540 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata/display.m',
'display', '1.12', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata',
'd8f1559d0bbcb161c8e9575263f0cd9', 'Output', 'xydata'; ...
0541 'eq', 'Relational Operator', 'xydata'; ...
0542 'findobj', 'xydata'; ...
0543 'findprop', 'xydata'; ...
0544 'ge', 'xydata'; ...
0545 'getDefaultPlist', 'xydata'; ...
0546 'getInfo', 'xydata'; ...
0547 'getX', 'Internal', 'xydata'; ...
0548 'getY', 'Internal', 'xydata'; ...
0549 'gt', 'xydata'; ...
0550 'isprop', 'Helper', 'xydata'; ...
0551 'isvalid', 'xydata'; ...
0552 'le', 'xydata'; ...
0553 'lt', 'xydata'; ...
0554 'ne', 'Relational Operator', 'xydata'; ...
0555 'notify', 'xydata'; ...
0556 'setX', 'Internal', 'xydata'; ...
0557 'setXY', 'Internal', 'xydata'; ...
0558 'setXunits', 'Internal', 'xydata'; ...
0559 'setY', 'Internal', 'xydata'; ...
0560 'setYunits', 'Internal', 'xydata'; ...
0561 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata/update_struct.m',
'update_struct', '1.2', '2008/09/05', '14:16:11', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata',
'bd84ab05b5cdedc0854e0c83365cb7ec', 'xydata'; ...
0562 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata/xydata.m',
'xydata', '1.32', '2008/09/03', '16:56:21', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xydata',
'1f19c11d4bf2b1e85ccb38244f659e2d', 'Constructor', 'xydata'; ...
0563 'SETS', 'data3D'; ...
0564 'VEROUT', 'data3D'; ...
0565 'addlistener', 'data3D'; ...
0566 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D/data3D.m',
'applymethod', '1.8', '2008/09/08', '08:29:13', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D',
'96cf25c89b981231a5e3c0d5e17afe62', 'data3D'; ...
0567 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D/data3D.m',
'applyoperator', '1.8', '2008/09/08', '08:29:13', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D',
'96cf25c89b981231a5e3c0d5e17afe62', 'data3D'; ...
0568 'char', 'data3D'; ...
0569 'copy', 'data3D'; ...
0570 '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D/data3D.m',
'data3D', '1.8', '2008/09/08', '08:29:13', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D',
'96cf25c89b981231a5e3c0d5e17afe62', 'Constructor', 'data3D'; ...

```

```

0571      'delete' 'data3D'; ...
0572      'display' 'data3D'; ...
0573      'eq' 'data3D'; ...
0574      'findobj' 'data3D'; ...
0575      'findprop' 'data3D'; ...
0576      'ge' 'data3D'; ...
0577      'getDefaultPlist' 'data3D'; ...
0578      'getInfo' 'data3D'; ...
0579      'getX' 'data3D'; ...
0580      'getY' 'data3D'; ...
0581      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D/getZ.m',
'getZ', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D',
'f1587be95041cd5cba6b21e61d41a10b', 'data3D'; ...
0582      'gt' 'data3D'; ...
0583      'isprop' 'data3D'; ...
0584      'isvalid' 'data3D'; ...
0585      'le' 'data3D'; ...
0586      'lt' 'data3D'; ...
0587      'ne' 'data3D'; ...
0588      'notify' 'data3D'; ...
0589      'setX' 'data3D'; ...
0590      'setXY' 'data3D'; ...
0591      'setXunits' 'data3D'; ...
0592      'setY' 'data3D'; ...
0593      'setYunits' 'data3D'; ...
0594      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D/setZ.m',
'setZ', '1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D',
'8b66b3263174a948370506ea630d4aaf', 'data3D'; ...
0595      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D/setZunits.m', 'setZunits',
'1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D',
'd19b07b62eba0c9a778e8392dbbaa86c', 'data3D'; ...
0596      'update_struct' 'data3D'; ...
0597      'SETS' 'xyzdata'; ...
0598      'VEROUT' 'xyzdata'; ...
0599      'addlistener' 'xyzdata'; ...
0600      'applymethod' 'xyzdata'; ...
0601      'applyoperator' 'xyzdata'; ...
0602      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata/char.m',
'char', '1.2', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata',
'b5db5a80d2c641f14deaaba3fe7de257', 'Output', 'xyzdata'; ...
0603      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata/copy.m',
'copy', '1.6', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata',
'badd828e8ef631ab9b5acd7b54994202', 'xyzdata'; ...
0604      'delete' 'xyzdata'; ...
0605      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata/display.m',
'display', '1.7', '2008/09/04', '15:29:32', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata',
'14583b3dab2a245efed3cf11c67a4b22', 'Output', 'xyzdata'; ...
0606      'eq' 'Relational Operator', 'xyzdata'; ...
0607      'findobj' 'xyzdata'; ...
0608      'findprop' 'xyzdata'; ...
0609      'ge' 'xyzdata'; ...
0610      'getDefaultPlist' 'xyzdata'; ...
0611      'getInfo' 'xyzdata'; ...
0612      'getX' 'Internal', 'xyzdata'; ...
0613      'getY' 'Internal', 'xyzdata'; ...
0614      'getZ' 'Internal', 'xyzdata'; ...
0615      'gt' 'xyzdata'; ...
0616      'isprop' 'Helper', 'xyzdata'; ...
0617      'isvalid' 'xyzdata'; ...
0618      'le' 'xyzdata'; ...
0619      'lt' 'xyzdata'; ...
0620      'ne' 'Relational Operator', 'xyzdata'; ...
0621      'notify' 'xyzdata'; ...
0622      'setX' 'Internal', 'xyzdata'; ...
0623      'setXY' 'Internal', 'xyzdata'; ...
0624      'setXunits' 'Internal', 'xyzdata'; ...
0625      'setY' 'Internal', 'xyzdata'; ...
0626      'setYunits' 'Internal', 'xyzdata'; ...
0627      'setZ' 'Internal', 'xyzdata'; ...
0628      'setZunits' 'Internal', 'xyzdata'; ...
0629      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata/update_struct.m',
'update_struct', '1.1', '2008/07/25', '18:14:27', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata',
'd97cfc52d388e101521b530621f7d1f6', 'xyzdata'; ...
0630      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata/xyzdata.m',
'xyzdata', '1.12', '2008/09/03', '16:39:37', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@xyzdata',
'82be91ca66035e388340206d8f3472cf', 'Constructor', 'xyzdata'; ...

```



```
0631      'SETS' 'specwin'; ...
0632      'VEROUT' 'specwin'; ...
0633      'addlistener' 'specwin'; ...
0634      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/char.m',
      'char', '1.7', '2008/09/04', '15:29:31', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin',
      '8dfff646f8dee12989f11e713084cd3', 'Output', 'specwin'; ...
0635      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/copy.m',
      'copy', '1.7', '2008/09/04', '16:29:33', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin',
      'd03eb8a7f4207beecb119c014ba23b89', 'specwin'; ...
0636      'delete' 'specwin'; ...
0637      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/display.m',
      'display', '1.10', '2008/09/04', '15:29:31', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin',
      '0cc0764e39df88526cf58cd5545c1949', 'Output', 'specwin'; ...
0638      'eq' 'Relational Operator', 'specwin'; ...
0639      'findobj' 'specwin'; ...
0640      'findprop' 'specwin'; ...
0641      'ge' 'specwin'; ...
0642      'getDefaultPlist' 'specwin'; ...
0643      'getInfo' 'specwin'; ...
0644      'getTypes' 'specwin'; ...
0645      'gt' 'specwin'; ...
0646      'isprop' 'Helper', 'specwin'; ...
0647      'isvalid' 'specwin'; ...
0648      'le' 'specwin'; ...
0649      'lt' 'specwin'; ...
0650      'ne' 'Relational Operator', 'specwin'; ...
0651      'notify' 'specwin'; ...
0652      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/plot.m',
      'plot', '1.9', '2008/09/04', '15:29:31', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin',
      '47128b9aa040e61213acabdc2027396c', 'Output', 'specwin'; ...
0653      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/specwin.m',
      'specwin', '1.53', '2008/09/03', '16:39:05', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin',
      'd94ba03ab2295ea133e9ac8f6cad8384', 'Constructor', 'specwin'; ...
0654      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/string.m',
      'string', '1.8', '2008/09/04', '15:29:31', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin',
      'ebf7aeba30d697b5fefcd3e6ac6acf72', 'Output', 'specwin'; ...
0655      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/update_struct.m',
      'update_struct', '1.1', '2008/07/10', '12:06:50', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin',
      '070618d096bc073695498d478de23e10', 'specwin'; ...
0656      'SETS' 'ltpda_filter'; ...
0657      'VEROUT' 'ltpda_filter'; ...
0658      'addHistory' 'ltpda_filter'; ...
0659      'addlistener' 'ltpda_filter'; ...
0660      'char' 'ltpda_filter'; ...
0661      'copy' 'ltpda_filter'; ...
0662      'delete' 'ltpda_filter'; ...
0663      'display' 'ltpda_filter'; ...
0664      'eq' 'ltpda_filter'; ...
0665      'findobj' 'ltpda_filter'; ...
0666      'findprop' 'ltpda_filter'; ...
0667      'ge' 'ltpda_filter'; ...
0668      'getDefaultPlist' 'ltpda_filter'; ...
0669      'getInfo' 'ltpda_filter'; ...
0670      'gt' 'ltpda_filter'; ...
0671      'isprop' 'ltpda_filter'; ...
0672      'isvalid' 'ltpda_filter'; ...
0673      'le' 'ltpda_filter'; ...
0674      'lt' 'ltpda_filter'; ...
0675      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter/ltpda_filter.m',
      'ltpda_filter', '1.14', '2008/09/03', '16:32:47', 'hewitson',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter',
      'c61bc581b2cacca32a62953a513d3e0d', 'Constructor', 'ltpda_filter'; ...
0676      'ne' 'ltpda_filter'; ...
0677      'notify' 'ltpda_filter'; ...
0678      'retrieve' 'ltpda_filter'; ...
0679      'save' 'ltpda_filter'; ...
0680      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter/setHistout.m',
      'setHistout', '1.3', '2008/09/04', '15:29:30', 'ingo',
      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter',
      '6f12c3130c2f4a0ef2f62b68a0d5af52', 'ltpda_filter'; ...
0681      'setName' 'ltpda_filter'; ...
0682      'string' 'ltpda_filter'; ...
0683      'submit' 'ltpda_filter'; ...
0684      'type' 'ltpda_filter'; ...
0685      'update_struct' 'ltpda_filter'; ...
0686      'SETS' 'mfir'; ...
```

```

0687      'VEROUT' 'Internal' 'mfir'; ...
0688      'addHistory' 'Internal' 'mfir'; ...
0689      'addlistener' 'mfir'; ...
0690      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/char.m',
'char', '1.7', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'b9a47cce9be5408270ff77c1cceede5', 'Output', 'mfir'; ...
0691      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'3911d5321f73c81739a5c4eab7a6382e', 'mfir'; ...
0692      'delete' 'mfir'; ...
0693      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/display.m',
'display', '1.12', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'ebe909e2af93676c4748820f18637c54', 'Output', 'mfir'; ...
0694      'eq' 'Relational Operator', 'mfir'; ...
0695      'findobj' 'mfir'; ...
0696      'findprop' 'mfir'; ...
0697      'ge' 'mfir'; ...
0698      'getDefaultPlist' 'mfir'; ...
0699      'getInfo' 'mfir'; ...
0700      'gt' 'mfir'; ...
0701      'isprop' 'Helper', 'mfir'; ...
0702      'isvalid' 'mfir'; ...
0703      'le' 'mfir'; ...
0704      'lt' 'mfir'; ...
0705      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/mfir.m',
'mfir', '1.63', '2008/09/07', '10:27:46', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'9948d90b82dab486f504709f04ecce18', 'Constructor', 'mfir'; ...
0706      'ne' 'Relational Operator', 'mfir'; ...
0707      'notify' 'mfir'; ...
0708      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/redesign.m',
'redesign', '1.10', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'1e00f4ff419c595c083cfc68be78fd4d', 'Helper', 'mfir'; ...
0709      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/resp.m',
'resp', '1.20', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'824f93418a14b5d2c73f741258225f96', 'Signal Processing', 'mfir'; ...
0710      'retrieve' 'mfir'; ...
0711      'save' 'Output', 'mfir'; ...
0712      'setHistout' 'Internal', 'mfir'; ...
0713      'setName' 'Helper', 'mfir'; ...
0714      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/string.m',
'string', '1.8', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'0602304e80d918c1e87253c370496361', 'Output', 'mfir'; ...
0715      'submit' 'Internal', 'mfir'; ...
0716      'type' 'Output', 'mfir'; ...
0717      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:49', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir',
'b428ad789bdc1d9c7252f9edc3156d3a', 'mfir'; ...
0718      'SETS' 'miir'; ...
0719      'VEROUT' 'miir'; ...
0720      'addHistory' 'Internal', 'miir'; ...
0721      'addlistener' 'miir'; ...
0722      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/char.m',
'char', '1.10', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'61646872f5d48b759e34fff03960f894', 'Output', 'miir'; ...
0723      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'e281b596a8b8610f7e03a1ccab70f503', 'miir'; ...
0724      'delete' 'miir'; ...
0725      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/display.m',
'display', '1.16', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'1707bf76e8c691a104c162cfa4eceb38', 'Output', 'miir'; ...
0726      'eq' 'Relational Operator', 'miir'; ...
0727      'findobj' 'miir'; ...
0728      'findprop' 'miir'; ...
0729      'ge' 'miir'; ...
0730      'getDefaultPlist' 'miir'; ...
0731      'getInfo' 'miir'; ...
0732      'gt' 'miir'; ...
0733      'isprop' 'Helper', 'miir'; ...
0734      'isvalid' 'miir'; ...
0735      'le' 'miir'; ...
0736      'lt' 'miir'; ...
0737      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/miir.m',
'miir', '1.75', '2008/09/07', '10:27:46', 'hewitson',

```



```

'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'5cea5339fc04d020665df4c41620e8fc', 'Constructor', 'miir'; ...
0738    'ne', 'Relational Operator', 'miir'; ...
0739    'notify', 'miir'; ...
0740    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/redesign.m',
'redesign', '1.15', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'c671e65cd1e7bc5dafc9d907b732ccb5', 'Helper', 'miir'; ...
0741    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/resp.m',
'resp', '1.20', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'a381c73ccb260f616ad939e0f33b6941', 'Signal Processing', 'miir'; ...
0742    'retrieve', 'miir'; ...
0743    'save', 'Output', 'miir'; ...
0744    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/setHistin.m',
'setHistin', '1.3', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'bc88d518fc1b6bea55756392a779c824', 'Internal', 'miir'; ...
0745    'setHistout', 'Internal', 'miir'; ...
0746    'setName', 'Helper', 'miir'; ...
0747    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/string.m',
'string', '1.9', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'bb7245348e6dc993e1fe0c615af295d8', 'Output', 'miir'; ...
0748    'submit', 'Internal', 'miir'; ...
0749    'type', 'Output', 'miir'; ...
0750    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:49', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir',
'da3c9db6eba2d6295dad11c71a5eb1a4', 'miir'; ...
0751    'SETS', 'pz'; ...
0752    'VEROUT', 'pz'; ...
0753    'addlistener', 'pz'; ...
0754    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/char.m', 'char',
'1.5', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'4c36a2f67ede087e2a5c13030861e76a', 'Output', 'pz'; ...
0755    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/copy.m', 'copy',
'1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'685325438d333248e38ae09a1119b614', 'pz'; ...
0756    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/cp2iir.m',
'cp2iir', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'eec126ff43cd4051699c593ff6fa2e89', 'Internal', 'pz'; ...
0757    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/cz2iir.m',
'cz2iir', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'a5dec880c1cb1988acc1e181ad19d554', 'Internal', 'pz'; ...
0758    'delete', 'pz'; ...
0759    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/display.m',
'display', '1.3', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'86d48841183effd0836be67e8b683529', 'Output', 'pz'; ...
0760    'eq', 'Relational Operator', 'pz'; ...
0761    'findobj', 'pz'; ...
0762    'findprop', 'pz'; ...
0763    'ge', 'pz'; ...
0764    'getDefaultPlist', 'pz'; ...
0765    'getInfo', 'pz'; ...
0766    'gt', 'pz'; ...
0767    'isprop', 'Helper', 'pz'; ...
0768    'isvalid', 'pz'; ...
0769    'le', 'pz'; ...
0770    'lt', 'pz'; ...
0771    'ne', 'Relational Operator', 'pz'; ...
0772    'notify', 'pz'; ...
0773    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/pz.m', 'pz',
'1.23', '2008/09/03', '16:36:02', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'd3664e4e5cc44dacaf28fb5343f41a02', 'Constructor', 'pz'; ...
0774    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/resp.m', 'resp',
'1.5', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'60a940b5afa432afd3728896a579d39a', 'Signal Processing', 'pz'; ...
0775    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/rp2iir.m',
'rp2iir', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'fdfb9313755842aa818f755738564309', 'Internal', 'pz'; ...
0776    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/rz2iir.m',
'rz2iir', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'8bb2db2719d4b70aef80a4adb9bcfd36', 'Internal', 'pz'; ...
0777    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/string.m',
'string', '1.4', '2008/09/04', '15:29:31', 'ingo',

```

```

'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'871a75d0da498bcfa14d9ada57c9d992', 'Output', 'pz'; ...
0778
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz/update_struct.m',
'update_struct', '1.2', '2008/07/11', '17:34:47', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pz',
'0232e21c0a4d6a1ed3309e89a532b197', 'pz'; ...
0779
    'SETS', 'pzmodel'; ...
0780
    'VEROUT', 'pzmodel'; ...
0781
    'addHistory', 'Internal', 'pzmodel'; ...
0782
    'addlistener', 'pzmodel'; ...
0783
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/char.m',
'char', '1.8', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'5e68b27bfa20ec2e27d8d53e57f0daee', 'Output', 'pzmodel'; ...
0784
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'da62cd458d6b799624764237806f955d', 'pzmodel'; ...
0785
    'delete', 'pzmodel'; ...
0786
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/display.m',
'display', '1.9', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'90a7e1b396801d1eae30ab1b2c36b6aa', 'Output', 'pzmodel'; ...
0787
    'eq', 'Relational Operator', 'pzmodel'; ...
0788
    'findobj', 'pzmodel'; ...
0789
    'findprop', 'pzmodel'; ...
0790
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/fngen.m',
'fngen', '1.10', '2008/09/05', '14:17:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'd233a0c71e42f7969db2901b08857b13', 'Signal Processing', 'pzmodel'; ...
0791
    'ge', 'pzmodel'; ...
0792
    'getDefaultPlist', 'pzmodel'; ...
0793
    'getInfo', 'pzmodel'; ...
0794
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/getLowerFreq.m',
'getLowerFreq', '1.7', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'22699a6d7929815f2bc33e95a4828e7', 'Internal', 'pzmodel'; ...
0795
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/getUpperFreq.m',
'getUpperFreq', '1.7', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'7c1b95342aef8a31cb3d4bf5de0c4afa', 'Internal', 'pzmodel'; ...
0796
    'gt', 'pzmodel'; ...
0797
    'isprop', 'Helper', 'pzmodel'; ...
0798
    'isvalid', 'pzmodel'; ...
0799
    'le', 'pzmodel'; ...
0800
    'lt', 'pzmodel'; ...
0801
    'ne', 'Relational Operator', 'pzmodel'; ...
0802
    'notify', 'pzmodel'; ...
0803
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/pzm2ab.m',
'pzm2ab', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'6fa9167e16fd5be30516777c3487a67d', 'Internal', 'pzmodel'; ...
0804
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/pzmodel.m',
'pzmodel', '1.60', '2008/09/07', '10:48:46', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'f52e2625938d9455c5aec1d33a2244d', 'Constructor', 'pzmodel'; ...
0805
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/resp.m',
'resp', '1.26', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'53cd471de2ff1eaa77a87bc4dbd529a9', 'Signal Processing', 'pzmodel'; ...
0806
    'retrieve', 'pzmodel'; ...
0807
    'save', 'Output', 'pzmodel'; ...
0808
    'setName', 'Helper', 'pzmodel'; ...
0809
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/string.m',
'string', '1.10', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'd0d272ac66951795a7a1d2c8d19be7c0', 'Helper', 'pzmodel'; ...
0810
    'submit', 'Internal', 'pzmodel'; ...
0811
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/tomfir.m',
'tomfir', '1.7', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'4fc35cb653ef575a182e473b5bc3d247', 'Operator', 'pzmodel'; ...
0812
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/tomiir.m',
'tomiir', '1.8', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'799e0b460bf099ef70cdbbc7b21051a6', 'Operator', 'pzmodel'; ...
0813
    'type', 'Output', 'pzmodel'; ...
0814
    '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:50', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@pzmodel',
'e8671342f48beb7c79799ee6a4abea25', 'pzmodel'; ...
0815
    'SETS', 'provenance'; ...

```

```

0816      'VEROUT', 'provenance'; ...
0817      'addlistener', 'provenance'; ...
0818      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance/char.m',
'char', '1.10', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance',
'fef0179f5f5b27be5d9fa56bc7e66667', 'Output', 'provenance'; ...
0819      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance',
'e370993a10b37db37c2cdd8e634f61be', 'provenance'; ...
0820      'delete', 'provenance'; ...
0821      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance/display.m', 'display',
'1.12', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance',
'63ee3b6b12d2fa8571706d64f9ced3da', 'Output', 'provenance'; ...
0822      'eq', 'Relational Operator', 'provenance'; ...
0823      'findobj', 'provenance'; ...
0824      'findprop', 'provenance'; ...
0825      'ge', 'provenance'; ...
0826      'getDefaultPlist', 'provenance'; ...
0827      'getInfo', 'provenance'; ...
0828      'gt', 'provenance'; ...
0829      'isprop', 'Helper', 'provenance'; ...
0830      'isvalid', 'provenance'; ...
0831      'le', 'provenance'; ...
0832      'lt', 'provenance'; ...
0833      'ne', 'Relational Operator', 'provenance'; ...
0834      'notify', 'provenance'; ...
0835      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance/provenance.m',
'provenance', '1.40', '2008/09/03', '16:35:40', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance',
'617ca06852ad0a20ee2f20df3ee948a1', 'Constructor', 'provenance'; ...
0836      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance/string.m', 'string',
'1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance',
'b96dcd6c0bf5f5843badb423a244e9be', 'Output', 'provenance'; ...
0837      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance/update_struct.m',
'update_struct', '1.4', '2008/08/22', '14:17:26', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@provenance',
'f4b5c64fbe093367204704d235b41f5e', 'provenance'; ...
0838      'SETS', 'time'; ...
0839      'VEROUT', 'time'; ...
0840      'addlistener', 'time'; ...
0841      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/char.m',
'char', '1.12', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'c47465c8ffbb8dc581a4e157b3a0df51', 'Output', 'time'; ...
0842      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'6b8fc575ce348090c8bf33936e4f8fd4', 'time'; ...
0843      'delete', 'time'; ...
0844      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/display.m',
'display', '1.19', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'2eed7e8f88513ae54f720688c91b8116', 'Output', 'time'; ...
0845      'eq', 'Relational Operator', 'time'; ...
0846      'findobj', 'time'; ...
0847      'findprop', 'time'; ...
0848      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/format.m',
'format', '1.11', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'4e97207899836692044ce8bffe2b0082', 'Output', 'time'; ...
0849      'ge', 'time'; ...
0850      'getDefaultPlist', 'time'; ...
0851      'getInfo', 'time'; ...
0852      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/getTimezones.m',
'getTimezones', '1.2', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'f737881d5adb1596d604973d717d9536', 'Helper', 'time'; ...
0853      'gt', 'time'; ...
0854      'isprop', 'Helper', 'time'; ...
0855      'isvalid', 'time'; ...
0856      'le', 'time'; ...
0857      'lt', 'time'; ...
0858      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/minus.m',
'minus', '1.12', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'ab1515c811f554ef30cd5008f621ce1f', 'Arithmetic Operator', 'time'; ...
0859      'ne', 'Relational Operator', 'time'; ...
0860      'notify', 'time'; ...

```



```

0861      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/plus.m',
'plus', '1.15', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'd5e609f7439625bc55fd04b4b0684655', 'Arithmetic Operator', 'time'; ...
0862      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/setEpochtime.m',
'setEpochtime', '1.5', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'147dfb55527f79ea1746858b2e6e323d', 'Internal', 'time'; ...
0863      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/setTime_str.m',
'setTime_str', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'c9e16df5ac22858b1887f4f022ed4187', 'Internal', 'time'; ...
0864      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/setTimeformat.m',
'setTimeformat', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'f2326e2520736d5d67be1dd4c1e6ce39', 'Internal', 'time'; ...
0865      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/setTimezone.m',
'setTimezone', '1.3', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'd1eda9244968f3823a65ad8e9dee86c3', 'Internal', 'time'; ...
0866      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/string.m',
'string', '1.9', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'dddf1b9cad67016a1e3d964d526bf187', 'Helper', 'time'; ...
0867      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/time.m',
'time', '1.56', '2008/09/03', '16:39:15', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'8eabb2b0e4c032f0571e79223b3a8079', 'Constructor', 'time'; ...
0868      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:50', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@time',
'0fc21796669a3b538442b991b9b2ec97', 'Internal', 'time'; ...
0869      'SETS', 'timespan'; ...
0870      'VEROUT', 'timespan'; ...
0871      'addHistory', 'Internal', 'timespan'; ...
0872      'addlistener', 'timespan'; ...
0873      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/char.m',
'char', '1.5', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'30232aa9f5a7978ef6fc9fe78c336cc7', 'Output', 'timespan'; ...
0874      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/copy.m',
'copy', '1.6', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'b59494c880f58aa7511ece19ff97510e', 'timespan'; ...
0875      'delete', 'timespan'; ...
0876      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/display.m', 'display',
'1.9', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'929dce913f0acc0cc602bf571d24adc1', 'Output', 'timespan'; ...
0877      'eq', 'Relational Operator', 'timespan'; ...
0878      'findobj', 'timespan'; ...
0879      'findprop', 'timespan'; ...
0880      'ge', 'timespan'; ...
0881      'getDefaultPlist', 'timespan'; ...
0882      'getInfo', 'timespan'; ...
0883      'gt', 'timespan'; ...
0884      'isprop', 'Helper', 'timespan'; ...
0885      'isvalid', 'timespan'; ...
0886      'le', 'timespan'; ...
0887      'lt', 'timespan'; ...
0888      'ne', 'Relational Operator', 'timespan'; ...
0889      'notify', 'timespan'; ...
0890      'retrieve', 'timespan'; ...
0891      'save', 'Output', 'timespan'; ...
0892      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/setEndT.m', 'setEndT',
'1.2', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'00b7217ec73162af1a127958750b6cc9', 'Helper', 'timespan'; ...
0893      'setName', 'Helper', 'timespan'; ...
0894      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/setStartT.m',
'setStartT', '1.2', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'ee940587e0e63d04405561f3f20e02fc', 'Helper', 'timespan'; ...
0895      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/setTimeformat.m',
'setTimeformat', '1.2', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'bfd451437f9c0d59acbd56835d6bd58b', 'Helper', 'timespan'; ...

```

```

0896
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/setTimezone.m',
'setTimezone', '1.2', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'c889cca2c7dba1db03619b5b97d43e76', 'Helper', 'timespan'; ...
0897
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/string.m',
'string', '1.5', '2008/09/04', '15:29:31', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'bdcaa92e1939c798500f31de2254e454', 'Output', 'timespan'; ...
0898
'', 'submit', '', '', '', 'Internal', 'timespan'; ...
0899
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/timespan.m', 'timespan',
'1.42', '2008/09/07', '10:48:46', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'4a8f7004395730aa6e322b74c9ba0207', 'Constructor', 'timespan'; ...
0900
'', 'type', '', '', '', 'Output', 'timespan'; ...
0901
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:50', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@timespan',
'f10c9439ea9ffad01c40fe2d7bad70ad', 'timespan'; ...
0902
'', 'SETS', '', '', '', 'history'; ...
0903
'', 'VEROUT', '', '', '', 'history'; ...
0904
'', 'addListener', '', '', '', 'history'; ...
0905
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/char.m',
'char', '1.8', '2008/09/04', '15:29:30', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'823832fb6ceeb71b6734dad53d48acec', 'Output', 'history'; ...
0906
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'7ac5f435ee1df54d9264e9be6638422e', 'history'; ...
0907
'', 'delete', '', '', '', 'history'; ...
0908
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/display.m',
'display', '1.16', '2008/09/04', '15:29:30', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'7a559d893c6e38ff15a9949a440a0660', 'Output', 'history'; ...
0909
'', 'eq', '', '', '', 'Relational Operator', 'history'; ...
0910
'', 'findobj', '', '', '', 'history'; ...
0911
'', 'findprop', '', '', '', 'history'; ...
0912
'', 'ge', '', '', '', 'history'; ...
0913
'', 'getDefaultPlist', '', '', '', 'history'; ...
0914
'', 'getInfo', '', '', '', 'history'; ...
0915
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/getNodes.m', 'getNodes',
'1.19', '2008/09/04', '15:29:30', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'46b54efd227894c2da56f2e3787a39ce', 'Internal', 'history'; ...
0916
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/getNodes_plot.m',
'getNodes_plot', '1.9', '2008/09/04', '15:29:30', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'abede47fba494629628523835dd834cb', 'Internal', 'history'; ...
0917
'', 'gt', '', '', '', 'history'; ...
0918
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/hist2dot.m', 'hist2dot',
'1.6', '2008/09/04', '15:29:30', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'454aa909d226769d79080e7c99589126', 'Output', 'history'; ...
0919
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/hist2m.m',
'hist2m', '1.30', '2008/09/05', '14:17:16', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'5b3055ca8b91a5fa19ccef464817ee', 'Output', 'history'; ...
0920
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/history.m',
'history', '1.41', '2008/09/08', '08:30:40', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'e27f1a7fffb22d0d60a8a523500e400c8', 'Constructor', 'history'; ...
0921
'', 'isprop', '', '', '', 'Helper', 'history'; ...
0922
'', 'isvalid', '', '', '', 'history'; ...
0923
'', 'le', '', '', '', 'history'; ...
0924
'', 'lt', '', '', '', 'history'; ...
0925
'', 'ne', '', '', '', 'Relational Operator', 'history'; ...
0926
'', 'notify', '', '', '', 'history'; ...
0927
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/plot.m',
'plot', '1.27', '2008/09/05', '14:17:01', 'hewitson'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'4fd8fcf1389305901c111fbb57ba00db', 'Output', 'history'; ...
0928
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/string.m',
'string', '1.8', '2008/09/04', '15:29:30', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'c13ad959f0a66e334b94bd14cdb0be65', 'Output', 'history'; ...
0929
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:49', 'ingo'
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history',
'6e729fe84549bb690df77fbc5fac3184', 'history'; ...

```

```

0930      'SETS' 'ssm'; ...
0931      'VEROUT' 'ssm'; ...
0932      'addHistory' 'Internal' 'ssm'; ...
0933      'addlistener' 'ssm'; ...
0934      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/assemble.m',
'assemble', '1.2', '2008/02/20', '09:20:07', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'a331691b74c59b326ef6d43ff3986b09', 'Statespace', 'ssm'; ...
0935      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/cell_add.m',
'cell_add',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'5d8cf1f27de0d9cbf3ca5768638cdcbb', 'ssm'; ...
0936      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/cell_fusion.m',
'cell_fusion',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'0cf8a6cdc953e8ee2ff85a663be613a4', 'ssm'; ...
0937      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/cell_mult.m',
'cell_mult',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'870de6deecf2ca0997bcc68b14c8638f', 'ssm'; ...
0938      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/cell_recut.m',
'cell_recut',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'8109738d4932b7ebcfbf8b7fda2dd2c1', 'ssm'; ...
0939      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/cell_select.m',
'cell_select',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'dbe2d6620fb355ce55b1916c3831a9ea', 'ssm'; ...
0940      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/cellstrfind.m',
'cellstrfind',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'75ad0948f6567e82f4e013b8eb83643f', 'ssm'; ...
0941      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/char.m',
'char', '1.9', '2008/07/21', '13:57:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'3a44a17e85dfcd8b7b32ea5524d4b043', 'Output', 'ssm'; ...
0942      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/copy.m',
'copy', '1.10', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'326a45d12aff62ca4a34f79402a5bfd8', 'ssm'; ...
0943      'delete' 'ssm'; ...
0944      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/display.m',
'display', '1.1', '2008/08/06', '18:35:26', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'9024845cb2d4e22a47a9d2f34d2bcb6f', 'Output', 'ssm'; ...
0945      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/double.m',
'double', '1.6', '2008/06/08', '18:44:07', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'bb8cdcc0ed9168fba8b3f38b269402fc', 'STATESPACE', 'ssm'; ...
0946      'eq' 'Relational Operator', 'ssm'; ...
0947      'findobj' 'ssm'; ...
0948      'findprop' 'ssm'; ...
0949      'ge' 'ssm'; ...
0950      'getDefaultPlist' 'ssm'; ...
0951      'getInfo' 'ssm'; ...
0952      'gt' 'ssm'; ...
0953      'isprop' 'Helper', 'ssm'; ...
0954      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/isstable.m',
'isstable', '1.1', '2008/08/06', '18:35:26', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'4a081d97809c79ad6eab7e3f88c4589f', 'Output', 'ssm'; ...
0955      'isvalid' 'ssm'; ...
0956      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/kalman.m',
'kalman',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'e2b84b9f426a5e59ab66c0e3e607f9fb', 'STATESPACE', 'ssm'; ...
0957      'le' 'ssm'; ...
0958      'lt' 'ssm'; ...
0959      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/minreal.m',
'minreal', '1.6', '2008/06/08', '18:44:07', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'1b36399fdc989dc13c4ec73a23e4826f', 'STATESPACE', 'ssm'; ...
0960      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/modifparams.m',
'modifparams', '1.17', '2008/07/22', '10:22:38', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'd823076377591ab591e9cc195be034f2', 'Statespace', 'ssm'; ...
0961      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/modiftimestep.m',
'modiftimestep', '1.5', '2008/03/11', '16:52:56', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'1a5986b643498a0710212a6b64ebba5f', 'STATESPACE', 'ssm'; ...
0962      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/modify.m',
'modify', '1.17', '2008/07/22', '10:22:38', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'114a38a8fdce158f62623ec0870a4efd', 'Statespace', 'ssm'; ...
0963      'ne' 'Relational Operator', 'ssm'; ...
0964      'notify', 'ssm'; ...

```

```

0965      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/reduce.m',
'reduce', '1.17', '2008/07/22', '10:22:38', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'7b1b662a80385170c8a810e9b484bd96', 'STATESPACE', 'ssm'; ...
0966      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/reduce_model.m',
'reduce_model', '1.17', '2008/07/22', '10:22:38', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'ba3ba5b2c05b65dc9f2b48f6378ffd9a', 'Statespace', 'ssm'; ...
0967      'retrieve', 'ssm'; ...
0968      'save', 'Output', 'ssm'; ...
0969      'setName', 'Helper', 'ssm'; ...
0970      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/simulate.m',
'simulate', 'ssm'; ...
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'acacd75d381e0d2a709ee13ca93b7ac6', 'STATESPACE', 'ssm'; ...
0971      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/ssm.m', 'ssm',
'1.67', '2008/09/07', '18:16:13', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'c0a542484dde4d782d375e550d40307', 'Constructor', 'ssm'; ...
0972      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/ssm2iirpz.m',
'ssm2iirpz', '1.3', '2008/04/21', '12:38:57', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'cc132a227c78329135f37d70a4066023', 'Statespace', 'ssm'; ...
0973      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/ssm2ss.m',
'ssm2ss', '1.18', '2008/08/26', '20:55:27', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'dc68c41524f7a0463fea6a6d9d9b0438', 'STATESPACE', 'ssm'; ...
0974      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/ssmFromss.m',
'ssmFromss', '1.7', '2008/06/13', '22:36:18', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'6e29206085e1029a5b57bf99fd04904b', 'ssm'; ...
0975      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/string.m',
'string', '1.8', '2008/07/21', '13:57:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'40d14b678297bcf37d6c279cb6fc3204', 'Output', 'ssm'; ...
0976      'submit', 'Internal', 'ssm'; ...
0977      'type', 'Output', 'ssm'; ...
0978      '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/update_struct.m',
'update_struct', '1.2', '2008/08/08', '15:32:33', 'adrien',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'7162d72ee54607c7e7060a76f8dd218c', 'ssm'; ...
0979      };

```


method_fcn_template

PURPOSE ^

FOO is the standard function template for class methods.

SYNOPSIS ^

function varargout = foo(varargin)

DESCRIPTION ^

```
FOO is the standard function template for class methods.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: FOO is the standard function template for class methods.
CALL:
    [a, b] = foo(a,pl)
INPUTS:
    a - analysis object(s)
    pl - parameter list(s)
        ...
OUTPUTS:
    a - ???
    b - ???
PARAMETER LIST:
<key>          <value>          <description>
'A'            'a'              parameter description
'N'            5                parameter description
DESCRIPTION    'User func. template' routine description
The following call returns a parameter list object that contains the
default parameter values:
>> pl = foo(ao, 'Params')
The following call returns a string that contains the routine CVS version:
>> version = foo(ao, 'Version')
The following call returns a string that contains the routine category:
>> category = foo(ao, 'Category')
VERSION:      $Id: method_fcn_template.m,v 1.6 2008/03/18 06:34:41 mauro Exp $
HISTORY:      dd-mm-yyyy M Hewitson
              Creation
SEE ALSO: ao/cos, ao/tan, ao/asin, acos, atan
SEE: <a href="matlab: help ao/Contents;">HELP: All Analysis Object Functions:</a>
     <a href="matlab: helpwin ao/Contents;">DOC: All Analysis Object Functions:</a>
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

 function plo = getDefaultPL()

SOURCE CODE ^

```

0001 function varargout = foo(varargin)
0002 % F00 is the standard function template for class methods.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: F00 is the standard function template for class methods.
0007 %
0008 % CALL:          foo(a)
0009 %              [a, b] = foo(a,pl)
0010 %
0011 % INPUTS:        a - analysis object(s)
0012 %                pl - parameter list(s)
0013 %                ...
0014 %
0015 % OUTPUTS:       a - ???
0016 %                b - ???
0017 %
0018 % PARAMETER LIST:
0019 % <key>          <value>          <description>
0020 % 'A'            'a'              parameter description
0021 % 'N'            5                parameter description
0022 % DESCRIPTION    'User func. template' routine description
0023 %
0024 % The following call returns a parameter list object that contains the
0025 % default parameter values:
0026 %
0027 % >> pl = foo(ao, 'Params')
0028 %
0029 % The following call returns a string that contains the routine CVS version:
0030 %
0031 % >> version = foo(ao, 'Version')
0032 %
0033 % The following call returns a string that contains the routine category:
0034 %
0035 % >> category = foo(ao, 'Category')
0036 %
0037 % VERSION:       $Id: method_fcn_template.m,v 1.6 2008/03/18 06:34:41 mauro Exp $
0038 %
0039 % HISTORY:       dd-mm-yyyy M Hewitson
0040 %               Creation
0041 %
0042 % SEE ALSO: ao/cos, ao/tan, ao/asin, acos, atan
0043 %
0044 % SEE: <a href="matlab: help ao/Contents;">HELP: All Analysis Object Functions:</a>
0045 %      <a href="matlab: helpwin ao/Contents;">DOC: All Analysis Object Functions:</a>
0046 %
0047 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0048 %
0049 ALGONAME = mfilename;
0050 VERSION = '$Id: method_fcn_template.m,v 1.6 2008/03/18 06:34:41 mauro Exp $';
0051 CATEGORY = 'Signal Processing';
0052 %
0053 %
0054 %% Check if this is a call for parameters, the CVS version string
0055 % or the method category
0056 %
0057 % Version for standard usage
0058 if nargin == 2 && ...
0059     isa(varargin{1}, 'ao') && ...
0060     ischar(varargin{2})
0061     in = char(varargin{2});
0062     if strcmpi(in, 'Params')
0063         varargout{1} = getDefaultPL();
0064         return
0065     elseif strcmpi(in, 'Version')
0066         varargout{1} = VERSION;
0067         return
0068     elseif strcmp(in, 'Category')
0069         varargout{1} = CATEGORY;
0070         return
0071     end
0072 end
0073 %
0074 %
0075 % Version for usage with sets of parameters, such as the class creators
0076 % if (nargin == 2 || nargin == 3) && ...
0077 %     isa(varargin{1}, 'ao') && ...
0078 %     ischar(varargin{2})
0079 %     in = char(varargin{2});
0080 %     if strcmp(in, 'Params')
0081 %         if nargin == 2

```

```

0082 %         varargout{1} = getDefaultPL();
0083 %     else
0084 %         varargout{1} = getDefaultPL(varargin{3});
0085 %     end
0086 %     return
0087 % elseif strcmp(in, 'Version')
0088 %     varargout{1} = VERSION;
0089 %     return
0090 % elseif strcmp(in, 'Category')
0091 %     varargout{1} = CATEGORY;
0092 %     return
0093 % end
0094 % end
0095
0096
0097 %% Collect input ao's, plist's and ao variable names
0098 in_names = {};
0099 for ii = 1:nargin
0100     in_names{end+1} = inputname(ii);
0101 end
0102
0103 [as, pl, invars] = collect_inputs(varargin, in_names);
0104
0105 % produce one parameter list
0106 if isa(pl, 'plist')
0107     pl = combine(pl, getDefaultPL());
0108 else
0109     pl = getDefaultPL();
0110 end
0111
0112 %% Get parameters
0113 A = find(pl, 'A');
0114 X = find(pl, 'X');
0115
0116
0117 %% Initialize outputs
0118 bs = [];
0119
0120
0121 %% Go through analysis objects
0122 for j=1: numel(as)
0123     a = as(j);
0124
0125     % get data
0126     [x,y] = get_xy_values(a.data);
0127
0128     % Do some analysis
0129     y = 2*A * (y + X);
0130     x = 1:length(x);
0131
0132     %%----- Build output AOs
0133
0134     % make a new xydata object, for example
0135     xy = xydata(x, y);
0136     xy = set(xy, 'name', sprintf('foo(%s)', a.data.name));
0137     xy = set(xy, 'xunits', a.data.xunits);
0138     xy = set(xy, 'yunits', 'Blah');
0139
0140     % make a new history object
0141     h = history(ALGONAME, VERSION, pl, a.hist);
0142     h = set(h, 'invars', invars);
0143
0144     % make output analysis object
0145     b = ao(xy, h);
0146
0147     % name, mfilename, description for this object
0148     b = setnh(b, 'name', sprintf('foo(%s)', invars{j}), ...
0149         'description', find(pl, 'description'));
0150
0151     bs = [bs b];
0152 end
0153
0154 end
0155
0156 %% Reshape the ouput to the same size of the input
0157 varargout{1} = reshape(bs, size(as));
0158
0159
0160
0161 %% %%%%%%%%%%%
0162 %
0163 % FUNCTION:     getDefaultPL
0164 %
0165 % DESCRIPTION:  Get default params, version for standard usage
0166 %
0167 % HISTORY:      dd-mm-yyyy M Hewitson

```

```
0168 %                               Creation
0169 %
0170 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0171 function plo = getDefaultPL()
0172
0173 plo = plist('A', 10, ...
0174 'X', [1], ...
0175 'DESCRIPTION', 'LTPDA Method Template');
0176
0177
0178
0179
0180 %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0181 %
0182 % FUNCTION:      getDefaultPL
0183 %
0184 % DESCRIPTION:  Get default params, version for usage with sets
0185 %
0186 % HISTORY:      dd-mm-yyyy M Hewitson
0187 %               Creation
0188 %
0189 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0190 % function out = getDefaultPL(varargin)
0191 %
0192 % % list of available parameter sets
0193 % sets = {'From XML file', 'From ASCII file', 'Function', 'Values',...
0194 %         'Time-series Function', 'From Window', 'Waveform'};
0195 %
0196 % if nargin == 0
0197 %     out = sets;
0198 %     return
0199 % end
0200 %
0201 % set = varargin{1};
0202 %
0203 % switch set
0204 %     case 'From XML file'
0205 %         out = plist('filename', 'foo.xml');
0206 %     case 'From ASCII file'
0207 %         out = plist('filename', 'foo.txt',...
0208 %                     'type', 'tsdata',...
0209 %                     'columns', [1 2],...
0210 %                     'num_columns', 10,...
0211 %                     'comment_char', '%');
0212 %     case 'Function'
0213 %         out = plist('fcn', 'randn(100,1)');
0214 %     case 'Values'
0215 %         out = plist('vals', [1 2 3]);
0216 %     case 'Time-series Function'
0217 %         out = plist('tsfcn', 'randn(size(t))', 'fs', 10, 'nsecs', 100);
0218 %     case 'From Window'
0219 %         out = plist('win', specwin('Hanning', 100));
0220 %     case 'Waveform'
0221 %         out = plist('waveform', 'sine wave',...
0222 %                     'f', 1.23,...
0223 %                     'phi', 0,...
0224 %                     'fs', 10,...
0225 %                     'nsecs', 10);
0226 %     otherwise
0227 %         out = plist();
0228 % end
0229
0230 % END
```

Contents

PURPOSE ^

%% path: m/gui/@jcontrol %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/@jcontrol %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help m/gui/@jcontrol/ancestor">m/gui/@jcontrol/ancestor</a> - function
overloaded as JCONTROL method
    <a href="matlab:help m/gui/@jcontrol/close">m/gui/@jcontrol/close</a> - methods
overloaded for JCONTROL objects
    <a href="matlab:help m/gui/@jcontrol/delete">m/gui/@jcontrol/delete</a> - method
overloaded for the JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/get">m/gui/@jcontrol/get</a> - method
overloaded for JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/getappdata">m/gui/@jcontrol/getappdata</a> -
function overloaded for JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/isappdata">m/gui/@jcontrol/isappdata</a> - function
overloaded for the JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/jcontrol">m/gui/@jcontrol/jcontrol</a> -
constructor for JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/rmappdata">m/gui/@jcontrol/rmappdata</a> - function
overloaded for the JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/set">m/gui/@jcontrol/set</a> - method
overloaded for JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/setappdata">m/gui/@jcontrol/setappdata</a> -
function overloaded for JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/subsasgn">m/gui/@jcontrol/subsasgn</a> - method
overloaded for JCONTROL class
    <a href="matlab:help m/gui/@jcontrol/subsref">m/gui/@jcontrol/subsref</a> - method
overloaded for jcontrol class
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

[Home](#) > [m](#) > [gui](#) > [@jcontrol](#) > ancestor.m

ancestor

PURPOSE [^]

ANCESTOR function overloaded as JCONTROL method

SYNOPSIS [^]

function h=ancestor(obj, type)

DESCRIPTION [^]

ANCESTOR function overloaded as JCONTROL method

Example:

h=ancestor(obj, type)

returns the appropriate ancestor of the obj.hgcontainer

See also builtin/ancestor

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ancestor](#) ANCESTOR function overloaded as JCONTROL method

This function is called by:

 [ancestor](#) ANCESTOR function overloaded as JCONTROL method

SOURCE CODE [^]

```
0001 function h=ancestor(obj, type)
0002 % ANCESTOR function overloaded as JCONTROL method
0003 %
0004 % Example:
0005 % h=ancestor(obj, type)
0006 %
0007 % returns the appropriate ancestor of the obj.hgcontainer
0008 %
0009 % See also builtin/ancestor
0010 %
0011
0012 h=ancestor(obj.hgcontainer, type);
0013 return
0014 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

close

PURPOSE ^

Close methods overloaded for JCONTROL objects

SYNOPSIS ^

function close(obj)

DESCRIPTION ^

```
Close methods overloaded for JCONTROL objects

Simply calls delete. Included as JCONTROL objects can sometimes be used
to replace figures/waitbars etc where existing code may cal close instead
of delete.

-----
Author: Malcolm Lidierth 07/07
Copyright The Author & King's College London 2007-
-----
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [delete](#) DELETE method overloaded for the JCONTROL class

This function is called by:

SOURCE CODE ^

```
0001 function close(obj)
0002 % Close methods overloaded for JCONTROL objects
0003 %
0004 % Simply calls delete. Included as JCONTROL objects can sometimes be used
0005 % to replace figures/waitbars etc where existing code may cal close instead
0006 % of delete.
0007 %
0008 % -----
0009 % Author: Malcolm Lidierth 07/07
0010 % Copyright The Author & King's College London 2007-
0011 % -----
0012
0013 delete(obj)
0014 return
0015 end
```


delete

PURPOSE ^

DELETE method overloaded for the JCONTROL class

SYNOPSIS ^

function delete(obj)

DESCRIPTION ^

```
DELETE method overloaded for the JCONTROL class

DELETE acts on the hgcontainer of the target object. Call this on the
parent JCONTROL to delete all its contents.

Example:
delete(obj)

-----
Author: Malcolm Lidierth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [delete](#) DELETE method overloaded for the JCONTROL class

This function is called by:

-  [close](#) Close methods overloaded for JCONTROL objects
-  [delete](#) DELETE method overloaded for the JCONTROL class

SOURCE CODE ^

```
0001 function delete(obj)
0002 % DELETE method overloaded for the JCONTROL class
0003 %
0004 % DELETE acts on the hgcontainer of the target object. Call this on the
0005 % parent JCONTROL to delete all its contents.
0006 %
0007 % Example:
0008 % delete(obj)
0009 %
0010 % -----
0011 % Author: Malcolm Lidierth 07/07
0012 % Copyright The Author & King's College London 2007
0013 % -----
0014 %
0015 delete(obj.hgcontainer);
0016 return
0017 end
```


[Home](#) > [m](#) > [gui](#) > [@jcontrol](#) > get.m

get

PURPOSE [^]

GET method overloaded for JCONTROL class

SYNOPSIS [^]

function s=get(obj, property)

DESCRIPTION [^]

GET method overloaded for JCONTROL class

Examples:

a=get(obj)

a=get(obj,PropertyName);

If obj is a vector of JCONTROLS, a cell array will be returned

See also: [jcontrol](#)

Author: Malcolm Lidierth 07/07

Copyright The Author & King's College London 2007

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [get](#) GET method overloaded for JCONTROL class

This function is called by:

 [get](#) GET method overloaded for JCONTROL class

SOURCE CODE [^]

```
0001 function s=get(obj, property)
0002 % GET method overloaded for JCONTROL class
0003 %
0004 % Examples:
0005 % a=get(obj)
0006 % a=get(obj,PropertyName);
0007 %
0008 % If obj is a vector of JCONTROLS, a cell array will be returned
0009 %
0010 % See also: jcontrol
0011 %
0012 % -----
0013 % Author: Malcolm Lidierth 07/07
0014 % Copyright The Author & King's College London 2007
0015 % -----
0016 %
0017 % No property: list contents
0018 if nargin==1
0019     if numel(obj)==1
0020         s.hgcontainer=obj.hgcontainer;
0021         s.hgcontrol=obj.hgcontrol;
0022         s.hghandle=obj.hghandle;
0023     else
0024         display(obj);
0025     end
0026     return
0027 end
```

```

0028
0029 % Must specify a single property
0030 if nargin>2
0031     error('Too many input arguments');
0032 end
0033
0034 % Ensure lower-case
0035 property=lower(property);
0036
0037 % Vector of JCONTROLS on input
0038 % Call GET on each in turn returning a cell array
0039 if numel(obj)>1
0040     if nargout>0
0041         % Pre-allocate
0042         s=cell(1,numel(obj));
0043         for idx=1:numel(obj)
0044             % Recursive call
0045             s{idx}=get(obj(idx),property);
0046         end
0047         return
0048     else
0049         % Stop problems in nested gets e.g. (get(get,...)... )
0050         error('Must specify a left-hand side assignment with vector input');
0051     end
0052 end
0053
0054 % Check for ambiguity in property name
0055 if isprop(obj.hgcontainer, property) &&...
0056     isprop(obj.hgcontrol, property)
0057     if strcmp(property,'visible')
0058         % Visible is an exception - take this from the container, MATLAB
0059         % links this property for the container and object
0060         s=obj.hgcontainer.Visible;
0061         return
0062     else
0063         error('Shared property name ''%s''\nYou must explicitly specify the target
object',...
0064             property);
0065     end
0066 end
0067
0068 % Do the work
0069 switch property
0070     case 'hghandle'
0071         s=obj.hghandle;
0072     case 'hgcontainer'
0073         s=obj.hgcontainer;
0074     case 'hgcontrol'
0075         s=obj.hgcontrol;
0076     otherwise
0077         if isprop(obj.hgcontainer, property)
0078             % container property
0079             s=obj.hgcontainer.(property);
0080         elseif isprop(obj.hgcontrol, property)
0081             % control property
0082             s=obj.hgcontrol.(property);
0083         elseif isempty(obj.hghandle)
0084             % Empty -default object
0085             s=[];
0086         else
0087             % Error
0088             error('No such property');
0089         end
0090 end
0091
0092 end

```

[Home](#) > [m](#) > [gui](#) > [@jcontrol](#) > getappdata.m

getappdata

PURPOSE [^]

GETAPPDATA function overloaded for JCONTROL class

SYNOPSIS [^]

function data=getappdata(obj, name)

DESCRIPTION [^]

```
GETAPPDATA function overloaded for JCONTROL class

GETAPPDATA returns data from the application data area of the hgcontrol
component of a JCONTROL

Example:
data=gatappdata(obj)
data=getappdata(obj, fieldName)

See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/ISAPPDATA
JCONTROL/RMAPPDATA GETAPPDATA

-----
Author: Malcolm Lidierth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

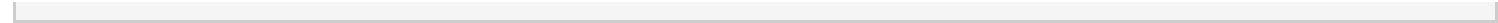
 [getappdata](#) GETAPPDATA function overloaded for JCONTROL class

This function is called by:

 [getappdata](#) GETAPPDATA function overloaded for JCONTROL class

SOURCE CODE [^]

```
0001 function data=getappdata(obj, name)
0002 % GETAPPDATA function overloaded for JCONTROL class
0003 %
0004 % GETAPPDATA returns data from the application data area of the hgcontrol
0005 % component of a JCONTROL
0006 %
0007 % Example:
0008 % data=gatappdata(obj)
0009 % data=getappdata(obj, fieldName)
0010 %
0011 % See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/ISAPPDATA
0012 % JCONTROL/RMAPPDATA GETAPPDATA
0013 %
0014 % -----
0015 % Author: Malcolm Lidierth 07/07
0016 % Copyright The Author & King's College London 2007
0017 % -----
0018
0019 if nargin==1
0020     data=getappdata(obj.hgcontrol);
0021 else
0022     data=getappdata(obj.hgcontrol, name);
0023 end
0024 return
0025 end
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

isappdata

PURPOSE ^

ISAPPPDATA function oveloaded for the JCONTROL class

SYNOPSIS ^

function isappdata(obj, name)

DESCRIPTION ^

```
ISAPPPDATA function oveloaded for the JCONTROL class

Examples:
isappdata(obj)
isappdata(obj, FieldName);

See also: JCONTROL, JCONTROL/GETAPPPDATA, JCONTROL/GETAPPPDATA
JCONTROL/ISAPPPDATA ISAPPPDATA

-----
Author: Malcolm Lidierth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [isappdata](#) ISAPPPDATA function oveloaded for the JCONTROL class

This function is called by:

 [isappdata](#) ISAPPPDATA function oveloaded for the JCONTROL class

SOURCE CODE ^

```
0001 function isappdata(obj, name)
0002 % ISAPPPDATA function oveloaded for the JCONTROL class
0003 %
0004 % Examples:
0005 % isappdata(obj)
0006 % isappdata(obj, FieldName);
0007 %
0008 % See also: JCONTROL, JCONTROL/GETAPPPDATA, JCONTROL/GETAPPPDATA
0009 % JCONTROL/ISAPPPDATA ISAPPPDATA
0010 %
0011 % -----
0012 % Author: Malcolm Lidierth 07/07
0013 % Copyright The Author & King's College London 2007
0014 % -----
0015
0016 isappdata(obj.hgcontrol, name);
0017 return
0018 end
```


[Home](#) > [m](#) > [gui](#) > [@jcontrol](#) > jcontrol.m

jcontrol

PURPOSE ^

JCONTROL constructor for JCONTROL class

SYNOPSIS ^

function obj=jcontrol(Parent, Style, varargin)

DESCRIPTION ^

JCONTROL constructor for JCONTROL class

JCONTROL provides an easy way to integrate a full range of java GUIs from the java.awt and javax.swing libraries into MATLAB.

Example:

```
obj=JCONTROL(Parent, Style);
obj=JCONTROL(Parent, Style, PropertyName1, PropertyValue1,...
    PropertyName2, PropertyValue2....);
```

Inputs:

Parent: the handle of a Matlab figure or other container for the resulting component

Style: string describing a java component e.g. 'javax.swing.JPanel', 'javax.swing.JButton' or a variable containing a java object

PropertyName/PropertyValue pairs: these are automatically assigned to the HG container or the java component as appropriate.

Pre-create the java object if you need to pass arguments to the constructor e.g.

```
javaobj=javax.swing...(.....);
obj=jcontrol(Parent, javaobj)
```

By default, JCONTROLS are returned with Units set to 'normalized'.

USE:

Build a GUI with repeated calls to JCONTROL in much the same way as with MATLAB's uicontrol function e.g.:

```
h=jcontrol(gcf,'javax.swing.JPanel',...
    'Position',[100 100 200 150],...
    'Units','normalized')
h(2)=jcontrol(h(1),'javax.swing.JComboBox',...
    'Position',[0.1 0.8 0.8 0.1]);
h(2).addItem('Item1');
h(2).addItem('Item2');
h(3)=jcontrol(h(1),'javax.swing.JCheckBox',...
    'Position',[0.1 0.1 0.1 0.1],...
    'Label','My check box');
```

See the jcontrolDemo() for a fuller example.

A JCONTROL aggregates the MATLAB handle graphics container and the Java component (as returned by MATLAB's JAVACOMPONENT function) into a single object.

Access to the JCONTROL's properties is provided by GET/SET calls. These automatically determine whether the target property is in the HG container or java object.

```
myobject=jcontrol(gcf,'javax.swing.JPanel',...
    'Units','normalized',...
    'Name','MyPanel');
set(myobject,'Position',[0.4 0.4 0.4 0.2],...
    'Enabled',0);
pos=get(myobject,'Units');
```

Note that you can mix HG container properties (e.g. Units, Position) and java component properties (e.g. Name, Enabled) in single calls to JCONTROL and SET.

Use the HG container to control the Units, Position, and Visible properties

MATLAB dot notation may also be used. This notation also provides access to the java object's methods

```
pos=myobject.Position;
sz=myObject.getSize;
myobject.setEnabled(1);
```

```
myobject.setToolTipText('My tip');
myobject.setOpaque(1);
```

UNITS, POSITION and VISIBLE properties

Set these by accessing the JCONTROL or its container (not the hgcontrol). MATLAB links these properties between the container and the java control, but unidirectionally. Note that JCONTROL methods always act on/return the Visible property of the container ('on' or 'off') which will also update the java control. Do not use the setVisible() methods.

Overloaded class methods are case-insensitive for properties but case-sensitive for java methods

CALLBACKS

Setting up callbacks

The simplest way to set up a callback is through the SET method

```
myPanel=jcontrol(gcf,'javax.swing.JPanel',...
    'Units','normalized',...
    'Position',[0.3 0.3 0.5 0.5]);
set(myPanel, 'MouseClickedCallback', 'MyCallback')
or
set(myPanel, 'MouseClickedCallback', @MyCallback);
or
set(myPanel, 'MouseClickedCallback', {@MyCallback A B C...});
```

The callback then takes the usual MATLAB form, e.g.

```
function MyCallback(hObject, eventdata)
function MyCallback(hObject, eventdata, varargin)
```

Accessing JCONTROL objects in callbacks:

The handle received by a callback will be that of the java control object contained in the JCONTROL, not the JCONTROL itself. In addition, GCO will return empty and GCBO will not return the parent figure handle. However, the JCONTROL constructor adds the HG container handle to the java component's properties. This can be used to access the container and its parent figure from within the callback e.g.

```
get(hObject.hghandle);% gets the HG container
ancestor(hObject.hghandle,'figure')% gets the parent figure handle
```

To cross-reference from the container, JCONTROL places a reference to the java control in the container's UserData area e.g.

```
hgc=findobj('Tag','MyCustomTag')
javacontrol=get(hgc, 'UserData');
```

Accessing data in callbacks

Data can be passed to a callback, as above, with optional input arguments. In addition, data that is specific to the control can be stored in the application data area of the control e.g. to return values dependent on the selection of a popup menu

```
data=getappdata(hObject,'data');
returnvalues=data(hObject.getSelecteditem+1);
```

Note: +1 because the item numbering is zero based for the java object. The HG container has a separate application data area.

R2006a or higher only:

GETAPPDATA, SETAPPDATA ISAPPDATA and RMAPPDATA methods have been overloaded for JCONTROL objects. These place/return data from the application data area of the java control. Take care if removing the whole application data area - TMW may place data in there too. The HG container has a separate application data area.

Notes:

If a property name occurs in both the HG container and the java object, the JCONTROL methods can not unambiguously identify the source/target and it must be defined explicitly by the user e.g.

```
get(myobject.hgcontainer,'Opaque');
set(myobject.hgcontrol, 'Opaque',0);
```

The JCONTROL methods test for ambiguity and issue an error message when it arises. Note that the test uses MATLAB's isprop and is case insensitive.

It may also detect properties not listed by the MATLAB builtin GET function for the hgcontrol such as Visible. The JCONTROL methods always act on the Visible property of the hgcontainer, letting MATLAB update the object automatically (see above).

The DeleteFcn property of the hgcontainer is set by the JAVACOMPONENT function. If this property is changed, the new callback must explicitly delete the hgcontrol.

See also:

```
jcontrol/get, jcontrol/set, jcontrol/subsasgn, jcontrol/subsref
jcontrol/setappdata, jcontrol/getappdata, jcontrol/isappdata
jcontrol/rmappdata, jcontrol/delete
javacomponent, gco, gcbo
```

 Acknowledgements: The JCONTROL class was partly inspired by Yair Altman's
<http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14583&objectType=file> uicomponent function.
 The functions take different routes to achieve similar ends. JCONTROL is rather
 faster - which is significant when building complex GUIs, but UICOMPONENT
 accepts the same calling conventions as MATLAB's UICONTROL while JCONTROL
 does not.

 Author: Malcolm Lidierth 07/07
 Copyright The Author & King's College London 2007-

Revisions:
 12.09.07 Allow pre-constructed java objects on input

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [set](#) SET method overloaded for JCONTROL class

This function is called by:

 [subsasgn](#) SUBSASGN method overloaded for JCONTROL class

SOURCE CODE [^]

```
0001 function obj=jcontrol(Parent, Style, varargin)
0002 % JCONTROL constructor for JCONTROL class
0003 %
0004 % JCONTROL provides an easy way to integrate a full range of java GUIs
0005 % from the java.awt and javax.swing libraries into MATLAB.
0006 %
0007 % Example:
0008 % obj=JCONTROL(Parent, Style);
0009 % obj=JCONTROL(Parent, Style, PropertyName1, PropertyValue1,...
0010 %              PropertyName2, ProeprtyValue2....);
0011 % Inputs:
0012 % Parent: the handle of a Matlab figure or other container for the resulting
0013 %         component
0014 % Style: string describing a java component e.g. 'javax.swing.JPanel',
0015 %        'javax.swing.JButton' or a variable containing a java object
0016 % PropertyName/PropertyValue pairs: these are automatically assigned to the
0017 %        HG container or the java component as appropriate.
0018 %
0019 % Pre-create the java object if you need to pass arguments to the
0020 % constructor e.g.
0021 % javaobj=javax.swing...(.....);
0022 % obj=jcontrol(Parent, javaobj)
0023 %
0024 % By default, JCONTROLS are returned with Units set to 'normalized'.
0025 %
0026 % USE:
0027 % Build a GUI with repeated calls to JCONTROL in much the same way as with
0028 % MATLAB's uicontrol function e.g.:
0029 %         h=jcontrol(gcf,'javax.swing.JPanel',...
0030 %                   'Position',[100 100 200 150],...
0031 %                   'Units','normalized')
0032 %         h(2)=jcontrol(h(1),'javax.swing.JComboBox',...
0033 %                   'Position',[0.1 0.8 0.8 0.1]);
0034 %         h(2).addItem('Item1');
0035 %         h(2).addItem('Item2');
0036 %         h(3)=jcontrol(h(1),'javax.swing.JCheckBox',...
0037 %                   'Position',[0.1 0.1 0.1 0.1],...
0038 %                   'Label','My check box');
0039 % See the jcontrolDemo() for a fuller example.
0040 %
0041 % A JCONTROL aggregates the MATLAB handle graphics container and the Java
0042 % component (as returned by MATLAB's JAVACOMPONENT function) into a single
0043 % object.
0044 % Access to the JCONTROL's properties is provided by GET/SET calls.
0045 % These automatically determine whether the target property is in
0046 % the HG container or java object.
0047 %         myobject=jcontrol(gcf,'javax.swing.JPanel',...
```

```

0048 %             'Units', 'normalized',...
0049 %             'Name', 'MyPanel');
0050 %         set(myobject, 'Position', [0.4 0.4 0.4 0.2],...
0051 %             'Enabled', 0);
0052 %         pos=get(myobject, 'Units');
0053 % Note that you can mix HG container properties (e.g. Units, Position) and
0054 % java component properties (e.g. Name, Enabled) in single calls to
0055 % JCONTROL and SET.
0056 % Use the HG container to control the Units, Position, and Visible
0057 % properties
0058 %
0059 % MATLAB dot notation may also be used. This notation also provides access
0060 % to the java object's methods
0061 %         pos=myobject.Position;
0062 %         sz=myObject.getSize;
0063 %         myobject.setEnabled(1);
0064 %         myobject.setToolTipText('My tip');
0065 %         myobject.setOpaque(1);
0066 %
0067 % -----
0068 % UNITS, POSITION and VISIBLE properties
0069 % Set these by accessing the JCONTROL or its container (not the hgcontrol).
0070 % MATLAB links these properties between the container and the java control,
0071 % but unidirectionally.
0072 % Note that JCONTROL methods always act on/return the Visible property of
0073 % the container ('on' or 'off') which will also update the java control.
0074 % Do not use the setVisible() methods.
0075 % -----
0076 %
0077 % Overloaded class methods are case-insensitive for properties but
0078 % case-sensitive for java methods
0079 %
0080 % CALLBACKS
0081 % Setting up callbacks
0082 % The simplest way to set up a callback is through the SET method
0083 %         myPanel=jcontrol(gcf, 'javax.swing.JPanel',...
0084 %             'Units', 'normalized',...
0085 %             'Position', [0.3 0.3 0.5 0.5]);
0086 %         set(myPanel, 'MouseClickedCallback', 'MyCallback')
0087 %         or
0088 %         set(myPanel, 'MouseClickedCallback', @MyCallback);
0089 %         or
0090 %         set(myPanel, 'MouseClickedCallback', {@MyCallback A B C...});
0091 %
0092 % The callback then takes the usual MATLAB form, e.g.
0093 %         function MyCallback(hObject, eventdata)
0094 %             function MyCallback(hObject, eventdata, varargin)
0095 %
0096 % Accessing JCONTROL objects in callbacks:
0097 % The handle received by a callback will be that of the java control
0098 % object contained in the JCONTROL, not the JCONTROL itself. In addition,
0099 % GCO will return empty and GCBO will not return the parent figure handle.
0100 % However, the JCONTROL constructor adds the HG container handle to the
0101 % java component's properties. This can be used to access the container and
0102 % its parent figure from within the callback e.g.
0103 %         get(hObject.hghandle); % gets the HG container
0104 %         ancestor(hObject.hghandle, 'figure') % gets the parent figure handle
0105 % To cross-reference from the container, JCONTROL places a reference to
0106 % the java control in the container's UserData area e.g.
0107 %         hgc=findobj('Tag', 'MyCustomTag')
0108 %         javacontrol=get(hgc, 'UserData');
0109 %
0110 % Accessing data in callbacks
0111 % Data can be passed to a callback, as above, with optional input
0112 % arguments. In addition, data that is specific to the control can be stored
0113 % in the application data area of the control e.g. to return values
0114 % dependent on the selection of a popup menu
0115 %         data=getappdata(hObject, 'data');
0116 %         returnvalues=data(hObject.getSelecteditem+1);
0117 % Note: +1 because the item numbering is zero based for the java object.
0118 % The HG container has a separate application data area.
0119 %
0120 % R2006a or higher only:
0121 % GETAPPDATA, SETAPPDATA ISAPPDATA and RMAPPDATA methods have been
0122 % overloaded for JCONTROL objects. These place/return data from the
0123 % application data area of the java control. Take care if removing the whole
0124 % application data area - TMW may place data in there too. The HG container
0125 % has a separate application data area.
0126 %
0127 % Notes:
0128 % If a property name occurs in both the HG container and the java object,
0129 % the JCONTROL methods can not unambiguously identify the source/target
0130 % and it must be defined explicitly by the user e.g.
0131 %         get(myobject.hgcontainer, 'Opaque');
0132 %         set(myobject.hgcontrol, 'Opaque', 0);
0133 % The JCONTROL methods test for ambiguity and issue an error message when

```

```

0134 % it arises. Note that the test uses MATLAB's isprop and is case
0135 % insensitive.
0136 % It may also detect properties not listed by the MATLAB builtin GET
0137 % function for the hgcontrol such as Visible. The JCONTROL methods always
0138 % act on the Visible property of the hgcontainer, letting MATLAB update the
0139 % object automatically (see above).
0140 %
0141 % The DeleteFcn property of the hgcontainer is set by the JAVACOMPONENT
0142 % function. If this property is changed, the new callback must explicitly
0143 % delete the hgcontrol.
0144 %
0145 % See also:
0146 %     jcontrol/get, jcontrol/set, jcontrol/subsasgn, jcontrol/subsref
0147 %     jcontrol/setappdata, jcontrol/getappdata, jcontrol/isappdata
0148 %     jcontrol/rmappdata, jcontrol/delete
0149 %     javacomponent, gco, gcbo
0150 %
0151 %-----
0152 % Acknowledgements: The JCONTROL class was partly inspired by Yair Altman's
0153 % <a href="http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?
0154 % objectId=14583&objectType=file">uicomponent</a> function.
0155 % The functions take different routes to achieve similar ends. JCONTROL is rather
0156 % faster - which is significant when building complex GUIs, but UICOMPONENT
0157 % accepts the same calling conventions as MATLAB's UICONTROL while JCONTROL
0158 % does not.
0159 %-----
0160 %
0161 % Author: Malcolm Lidiierth 07/07
0162 % Copyright The Author & King's College London 2007-
0163 %-----
0164 %
0165 % Revisions:
0166 %     12.09.07 Allow pre-constructed java objects on input
0167
0168 if nargin==0
0169     % Default constructor
0170     obj.hgcontrol=[];
0171     obj.hghandle=[];
0172     obj.hgcontainer=[];
0173     obj=class(orderfields(obj),'jcontrol');
0174     return
0175 end
0176
0177
0178 % Check parent
0179 if ishandle(Parent) && Parent==0
0180     % Root given as Parent-create a new figure
0181     container=figure('MenuBar','none');
0182 elseif strcmp(class(Parent),'jcontrol');
0183     % Parent is a jcontrol - use the hgcontainer
0184     container=Parent.hgcontainer;
0185 else
0186     % Use as specified
0187     container=Parent;
0188 end
0189
0190 % Check the Parent is a valid handle
0191 if ishandle(container)==0
0192     error('Parent is not a valid handle (has the parent figure been closed?)');
0193 end
0194
0195 % Java object
0196 if ischar(Style)
0197     % Create a default object
0198     javaobject=javaObject(Style);
0199 elseif isjava(Style)
0200     % or use supplied object
0201     javaobject=Style;
0202 end
0203
0204 % Check we have a valid Style
0205 if isa(javaobject,'java.awt.Window')
0206     error('%s is a top-level container: \n%s\n',...
0207         Style,'it can not have a MATLAB figure as parent/ancestor');
0208 end
0209
0210 % If so, add callbacks and place in container
0211 [obj.hgcontrol containerHandle]=javacomponent(javaobject,[],container);
0212 % Put a copy of the container handle in obj....
0213 obj.hghandle=containerHandle;
0214 % ... and in the control
0215 s=schema.prop(obj.hgcontrol,'hghandle','mxArray');
0216 obj.hgcontrol.hghandle=containerHandle;
0217 s.AccessFlags.PublicSet='off';
0218

```



```
0219 % Put the container in obj
0220 obj.hgcontainer = handle(containerHandle);
0221
0222 % Construct the instance
0223 obj=class(orderfields(obj), 'jcontrol');
0224
0225 % Put a reference to the hgcontrol in the UserData area of the handle
0226 set(containerHandle, 'UserData', obj.hgcontrol);
0227
0228 % Set the properties and return
0229 % Default to normalized
0230 set(obj, 'Units', 'normalized');
0231 % Set values as requested
0232 if ~isempty(varargin)
0233     set(obj, varargin{:});
0234     return
0235 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

rmappdata

PURPOSE ^

RMAPPDATA function oveloaded for the JCONTROL class

SYNOPSIS ^

function rmappdata(obj, name)

DESCRIPTION ^

```
RMAPPDATA function oveloaded for the JCONTROL class

Examples:
rmappdata(obj)
rmappdata(obj, FieldName);

See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/GETAPPDATA
JCONTROL/ISAPPDATA RMAPPDATA

-----
Author: Malcolm Lidierth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [rmappdata](#) RMAPPDATA function oveloaded for the JCONTROL class

This function is called by:

 [rmappdata](#) RMAPPDATA function oveloaded for the JCONTROL class

SOURCE CODE ^

```
0001 function rmappdata(obj, name)
0002 % RMAPPDATA function oveloaded for the JCONTROL class
0003 %
0004 % Examples:
0005 % rmappdata(obj)
0006 % rmappdata(obj, FieldName);
0007 %
0008 % See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/GETAPPDATA
0009 % JCONTROL/ISAPPDATA RMAPPDATA
0010 %
0011 % -----
0012 % Author: Malcolm Lidierth 07/07
0013 % Copyright The Author & King's College London 2007
0014 % -----
0015
0016 rmappdata(obj.hgcontrol, name);
0017 return
0018 end
```


set

PURPOSE ^

SET method overloaded for JCONTROL class

SYNOPSIS ^

function varargout=set(varargin)

DESCRIPTION ^

```
SET method overloaded for JCONTROL class

Examples:
set(obj,PropertyName, PropertyValue)
set(obj,PropertyName1, Value1, PropertyName2, Value2....)

The propertyname/valuenam sequence can have an embedded cell array if
that contains property/value pairs e.g
    standardvalues={'Units', 'normalized'}
    set(myobj, 'javax.swing.JPane',...
              'ToolTipText','MyTip',...
              standardvalues);

S
Also:
set(obj, s)
set(obj, pn, pm)
where s in a structure with fieldnames corresponding to property names
and values corresponding to property values
pn and pm and name/value cell vectors (note pm may not be a matrix)

If obj is a vector of JCONTROLS, SET will act on each element.
In this case, the specified properties must be present in all the
JCONTROLS or an error will result

See also: jcontrol

-----
Author: Malcolm Lidierrth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [set](#) SET method overloaded for JCONTROL class

This function is called by:

-  [jcontrol](#) JCONTROL constructor for JCONTROL class
-  [set](#) SET method overloaded for JCONTROL class

SOURCE CODE ^

```
0001 function varargout=set(varargin)
0002 % SET method overloaded for JCONTROL class
0003 %
0004 % Examples:
0005 % set(obj,PropertyName, PropertyValue)
0006 % set(obj,PropertyName1, Value1, PropertyName2, Value2....)
0007 %
0008 % The propertyname/valuenam sequence can have an embedded cell array if
```

```

0009 % that contains property/value pairs e.g
0010 %         standardvalues={'Units', 'normalized'}
0011 %         set(myobj, 'javax.swing.JPane',...
0012 %         'ToolTipText','MyTip',...
0013 %         standardvalues);
0014 %s
0015 % Also:
0016 % set(obj, s)
0017 % set(obj, pn, pm)
0018 % where s is a structure with fieldnames corresponding to property names
0019 % and values corresponding to property values
0020 % pn and pm are name/value cell vectors (note pm may not be a matrix)
0021 %
0022 % If obj is a vector of JCONTROLS, SET will act on each element.
0023 % In this case, the specified properties must be present in all the
0024 % JCONTROLS or an error will result
0025 %
0026 %
0027 % See also: jcontrol
0028 %
0029 % -----
0030 % Author: Malcolm Lidierth 07/07
0031 % Copyright The Author & King's College London 2007
0032 % -----
0033
0034 % Note: The usual assignin(...) call is not needed here because both the
0035 % hgcontainer and the hgcontrol are passed by reference rather than by
0036 % value. Changes here affect the original JCONTROL contents, and all copies
0037 % of them in all MATLAB workspaces.
0038
0039 obj=varargin{1};
0040
0041 if nargin==1
0042     % Return settable properties
0043     s=set(obj.hgcontainer);
0044     s.hgcontrol=set(obj.hgcontrol);
0045     varargout{1}=s;
0046     return
0047 end
0048
0049 % Otherwise make sure we have enough inputs
0050 if nargin==2 && isstruct(varargin{2})
0051     % Property/value pairs in structure
0052     % Get the names
0053     propnames=fieldnames(varargin{2});
0054     % get the values
0055     propvalues=cell(length(propnames),1);
0056     for i=1:length(propnames)
0057         propvalues{i}=varargin{2}.(propnames{i});
0058     end
0059     % Recursive call to set
0060     set(varargin{1}, propnames, propvalues);
0061     return
0062 elseif nargin==3 && iscell(varargin{2}) && iscell(varargin{3})
0063     % set(obj, pn, pm) where pn and pm are property and value cell vectors
0064     if numel(varargin{2})~=numel(varargin{3})
0065         error('Multiple values for each property not supported'); %#ok<ERTAG>
0066     end
0067     proplist=cell(1,2*length(varargin{2}));
0068     count=1;
0069     for i=1:length(varargin{2})
0070         proplist(count)=varargin{2}(i);
0071         proplist(count+1)=varargin{3}(i);
0072         count=count+2;
0073     end
0074 else
0075     % Allow cell arrays in the property/values pairs list
0076     % Expand these where present
0077     pcount=1;
0078     vcount=2;
0079     proplist={};
0080     for i=1:(length(varargin)-1)/2
0081         if ischar(varargin{vcount})
0082             % Property/Value pair
0083             proplist{pcount}=varargin{vcount};
0084             proplist{pcount+1}=varargin{vcount+1};
0085             pcount=pcount+2;
0086             vcount=vcount+2;
0087         elseif iscell(varargin{vcount})
0088             % Cell array
0089             proplist=[proplist varargin{vcount}]; %#ok<AGROW>
0090             pcount=length(proplist)+1;
0091             vcount=vcount+1;
0092         end
0093     end
0094 end

```

```

0095
0096 % For loop allows vector (or matrix) of JCONTROLS on input but all elements
0097 % must have all the target properties or an error will be generated
0098 for idx=1:numel(obj)
0099     switch lower(proplist{1})
0100         case {'hgcontainer', 'hgcontrol', 'hghandle', 'Parent', 'Children'}
0101             error('The %s property of a jcontrol object can not be changed',
lower(varargin{2}));
0102         otherwise
0103             for i=1:2:length(proplist)
0104                 property=proplist{i};
0105                 value=proplist{i+1};
0106                 if isprop(obj(idx).hgcontainer, property) &&...
0107                     isprop(obj(idx).hgcontrol, property)
0108                     if strcmpi(property, 'Visible')
0109                         % Set Visible property in the container, MATLAB will
0110                         % update the hgcontrol
0111                         obj(idx).hgcontainer.Visible=VisibleProperty(value);
0112                         continue
0113                     else
0114                         error('Shared property name '%s'\nYou must explicitly specify
the target object',...
0115                             prop(i));
0116                     end
0117             end
0118             if isprop(obj(idx).hgcontainer, property)
0119                 obj(idx).hgcontainer.(property)=value;
0120             elseif isprop(obj(idx).hgcontrol, property)
0121                 obj(idx).hgcontrol.(property)=value;
0122             else
0123                 error('No such property: '%s' in %s(%d)', property, inputname(1),
idx);
0124             end
0125         end
0126     end
0127 end
0128 return
0129 end

```

[Home](#) > [m](#) > [gui](#) > [@jcontrol](#) > setappdata.m

setappdata

PURPOSE [^]

SETAPPDATA function overloaded for JCONTROL class

SYNOPSIS [^]

function setappdata(obj, name, data)

DESCRIPTION [^]

```
SETAPPDATA function overloaded for JCONTROL class

SETAPPDATA places data in the application data area of the hgcontrol
component of a JCONTROL
```

```
Example:
setappdata(obj, fieldname, data)
```

```
See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/ISAPPDATA
JCONTROL/RMAPPDATA SETAPPDATA
```

```
-----
Author: Malcolm Lidiierth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [setappdata](#) SETAPPDATA function overloaded for JCONTROL class

This function is called by:

 [setappdata](#) SETAPPDATA function overloaded for JCONTROL class

SOURCE CODE [^]

```
0001 function setappdata(obj, name, data)
0002 % SETAPPDATA function overloaded for JCONTROL class
0003 %
0004 % SETAPPDATA places data in the application data area of the hgcontrol
0005 % component of a JCONTROL
0006 %
0007 % Example:
0008 % setappdata(obj, fieldname, data)
0009 %
0010 % See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/ISAPPDATA
0011 % JCONTROL/RMAPPDATA SETAPPDATA
0012 %
0013 % -----
0014 % Author: Malcolm Lidiierth 07/07
0015 % Copyright The Author & King's College London 2007
0016 % -----
0017 %
0018 % Revisions:
0019 % 17.08.07 Check class of obj. This method gets called when data is a
0020 %           jcontrol object rather than obj
0021 %
0022 if strcmpi(class(obj), 'jcontrol')
0023     setappdata(obj.hgcontrol, name, data);
0024 else
0025     % Get here because data is a jcontrol object
0026     builtin('setappdata', obj, name, data)
```

```
0027 end
0028 return
0029 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [@jcontrol](#) > subsasgn.m

subsasgn

PURPOSE [^]

SUBSASGN method overloaded for JCONTROL class

SYNOPSIS [^]

function obj=subsasgn(obj, index, val)

DESCRIPTION [^]

```

SUBSASGN method overloaded for JCONTROL class

subsasgn provides access to JCONTROL properties via MATLAB's dot notation
Examples:
obj.Units='characters';
obj.Enabled=1
obj.hgcontainer.Opaque='on'

obj may be an element of a JCONTROL vector e.g.
obj(3).Units='pixels';

See also: jcontrol
-----
Author: Malcolm Lidierth 06/07
Copyright The Author & King's College London 2007
-----

```

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [jcontrol](#) JCONTROL constructor for JCONTROL class
-  [subsasgn](#) SUBSASGN method overloaded for JCONTROL class

This function is called by:

-  [subsasgn](#) SUBSASGN method overloaded for JCONTROL class

SOURCE CODE [^]

```

0001 function obj=subsasgn(obj, index, val)
0002 % SUBSASGN method overloaded for JCONTROL class
0003 %
0004 % subsasgn provides access to JCONTROL properties via MATLAB's dot notation
0005 % Examples:
0006 % obj.Units='characters';
0007 % obj.Enabled=1
0008 % obj.hgcontainer.Opaque='on'
0009 %
0010 % obj may be an element of a JCONTROL vector e.g.
0011 % obj(3).Units='pixels';
0012 %
0013 % See also: jcontrol
0014 % -----
0015 % Author: Malcolm Lidierth 06/07
0016 % Copyright The Author & King's College London 2007
0017 % -----
0018 %
0019 % Note: Left-hand assignments are not needed with dot assignments
0020 % because both the hgcontainer and the hgcontrol are passed by reference
0021 % rather than by value.
0022 % With () calls, left-hand assignments are needed when initializing an
0023 % element. In these cases the caller workspace will contain a reference to

```

```

0024 % the JCONTROL set up by subsasgn or passed to subsasgn by MATLAB in val.
0025 % In all cases, changes here to JCONTROL properties affect the original
0026 % JCONTROL contents, and all copies of them in all MATLAB workspaces.
0027
0028 switch index(1).type
0029     case '.'
0030         switch lower(index(1).subs)
0031             case {'hgcontainer' 'hgcontrol' 'hghandle'}
0032                 if length(index)==1 || strcmp(index(1).subs, 'hghandle')
0033                     error('The %s property is not settable', index(1).subs);
0034                 else
0035                     subsasgn(obj.(index(1).subs), index(2:end), val);
0036                 end
0037             otherwise
0038                 % obj.property/method where the property could be in hgcontainer
0039                 % or hgcontrol. Find out which and invoke appropriate subsasgn
0040                 if isprop(obj.hgcontainer, index(1).subs) &&...
0041                     isprop(obj.hgcontrol, index(1).subs)
0042                     if strcmpi(index(1).subs, 'visible')
0043                         % Set Visible property in the container, MATLAB will
0044                         % update the hgcontrol
0045                         obj.hgcontainer.Visible=VisibleProperty(val);
0046                         return
0047                     else
0048                         error('Shared property name '%s'\nYou must explicitly specify the
target object',...
                                index(1).subs);
0049                     end
0050                 end
0051             end
0052             if isprop(obj.hgcontainer, index(1).subs)
0053                 % hgcontainer property
0054                 subsasgn(obj.hgcontainer, index, val);
0055             elseif isprop(obj.hgcontrol, index(1).subs) ||
ismethod(obj.hgcontrol, index(1).subs)
0056                 % hgcontrol property or method
0057                 subsasgn(obj.hgcontrol, index, val);
0058             else
0059                 error('No such property or method');
0060             end
0061         end
0062     case '()'
0063         % obj is empty so initialize
0064         if isempty(obj)
0065             % Assign first element - avoids conversion to double errors
0066             % when initializing with subscripts
0067             obj=jcontrol();
0068             % Set the target element - need not already exist
0069             obj(index(1).subs{1})=val;
0070             return
0071         end
0072     if strcmp(class(val), 'jcontrol')
0073         % Initializing or adding element to an array
0074         obj(index(1).subs{1})=val;
0075     else
0076         % Assigning field to pre-existing element
0077         if isvector(obj) && length(index(1).subs)==1 &&...
                                index(1).subs{1}<=length(obj)
0078             subsasgn(obj(index(1).subs{1}), index(2:end), val);
0079         elseif ~isvector(obj) || length(index(1).subs)>1
0080             error('Scalar or vector JCONTROL and index required on input');
0081         elseif index(1).subs{1}>length(obj)
0082             error('Index exceeds length of JCONTROL vector');
0083         end
0084     end
0085 end
0086 end
0087 end
0088 return
0089 end
0090

```


subsref

PURPOSE ^

SUBSREF method overloaded for jcontrol class

SYNOPSIS ^

function ret=subsref(obj, index)

DESCRIPTION ^

```
SUBSREF method overloaded for jcontrol class

subsref provides access to jcontrol properties via MATLAB's dot notation
Examples:
obj.hgcontainer
obj.hgcontrol.Name

subsref also provides access to the java component's methods
Example:
obj.setToolTipText('MyText');
a=obj.getToolTipText();

See also: jcontrol

-----
Author: Malcolm Lidierth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [subsref](#) SUBSREF method overloaded for jcontrol class

This function is called by:

 [subsref](#) SUBSREF method overloaded for jcontrol class

SUBFUNCTIONS ^

 [function](#) CheckErr()

SOURCE CODE ^

```
0001 function ret=subsref(obj, index)
0002 % SUBSREF method overloaded for jcontrol class
0003 %
0004 % subsref provides access to jcontrol properties via MATLAB's dot notation
0005 % Examples:
0006 % obj.hgcontainer
0007 % obj.hgcontrol.Name
0008 %
0009 % subsref also provides access to the java component's methods
0010 % Example:
0011 % obj.setToolTipText('MyText');
0012 % a=obj.getToolTipText();
0013 %
0014 % See also: jcontrol
0015 %
0016 % -----
```

```

0017 % Author: Malcolm Lidieth 07/07
0018 % Copyright The Author & King's College London 2007
0019 % -----
0020
0021 % Revisions:
0022 % 21.08.07 Methods now called properly when index(1).subs=='hgcontrol'.
0023 % Previously subsref(obj.hgcontrol,...) was called from here with
0024 % index(1:end) instead of index(2:end).
0025 % 21.08.07 Tests nargout before calling hgcontrol methods. No output is
0026 % requested if nargout==0. This prevents "One or more output
0027 % arguments not assigned..." errors being generated internally
0028 % e.g calling setXXXX calls. These errors will still be generated
0029 % (as they should)if an output is requested when none is
0030 % available. See 09.09.07 below.
0031 % 09.09.07 Improve above fix with try/catch blocks. Now getXXXX calls
0032 % return in ans if nargout==0
0033 % 20.09.07 Nest try/catch blocks to reduce isprop() and ismethod()
0034 % calls. This is substantially faster.
0035
0036 switch index(1).type
0037     case '.'
0038         switch lower(index(1).subs)
0039             case 'hgcontainer'
0040                 if length(index)==1
0041                     % obj.hgcontainer
0042                     ret=obj.hgcontainer;
0043                 elseif isprop(obj.hgcontainer, index(2).subs)
0044                     % obj.hgcontainer.property
0045                     ret=subsref(obj.hgcontainer,index(2:end));
0046                 elseif isempty(obj.hgcontainer)
0047                     % Empty default object?
0048                     ret=[];
0049                 else
0050                     % Otherwise no property with this name
0051                     error('No appropriate property %s',index(2).subs);
0052             end
0053         case 'hgcontrol'
0054             if length(index)==1
0055                 % obj.hgcontrol
0056                 ret=obj.hgcontrol;
0057             else
0058                 try
0059                     ret=subsref(obj.hgcontrol, index(2:end));
0060                 catch
0061                     try
0062                         CheckErr();
0063                         subsref(obj.hgcontrol, index(2:end));
0064                     catch
0065                         if isempty(obj.hghandle)
0066                             % Maybe subsref failed because obj has
0067                             % empty properties
0068                             ret=[];
0069                         elseif ~isprop(obj.hghandle, index(2).subs) &&...
0070                             ~ismethod(obj.hghandle, index(2).subs)
0071                             error('jcontrol:subsref: No such property or method');
0072                         else
0073                             error('jcontrol:subsref: Unexpected error');
0074                         end
0075                     end
0076                 end
0077             end
0078         case 'hghandle'
0079             ret=obj.hghandle;
0080         otherwise
0081             % obj.property/method where the property could be in hgcontainer
0082             % or hgcontrol. Find out which and invoke subsref
0083             % recursively
0084             if isprop(obj.hgcontainer, index(1).subs) &&...
0085                 isprop(obj.hgcontrol, index(1).subs)
0086                 % Visible is an exception - take this from the container, MATLAB
0087                 % links this property for the container and object
0088                 if strcmpi(index(1).subs,'visible')
0089                     ret=obj.hgcontainer.Visible;
0090                     return
0091                 else
0092                     error('Shared property name '%s'\nYou must explicitly specify the
0093 target object',...
0094                             index(1).subs);
0095                 end
0096             end
0097             if isprop(obj.hgcontainer, index(1).subs)
0098                 % hgcontainer property
0099                 ret=subsref(obj.hgcontainer, index);
0100             else
0101                 % hgcontrol property or method?

```

```

0102         try
0103             ret=subsref(obj.hgcontrol, index);
0104         catch
0105             try
0106                 CheckErr();
0107                 subsref(obj.hgcontrol, index);
0108             catch
0109                 if isempty(obj.hghandle)
0110                     % Maybe subsref failed because obj has
0111                     % empty properties
0112                     ret=[];
0113                 elseif ~isprop(obj.hghandle, index(1).subs) &&...
0114                     ~ismethod(obj.hghandle, index(1).subs)
0115                     error('jcontrol:subsref: No such property or method');
0116                 else
0117                     error('jcontrol:subsref: Unexpected error');
0118                 end
0119             end
0120         end
0121     end
0122 end
0123 case '()'
0124     % array of jcontrols
0125     if length(index)==1
0126         % One or more elements or a JCONTROL array/matrix wanted
0127         ret=obj(index(1).subs{:});
0128     else
0129         obj=obj(index(1).subs{:});
0130         if numel(obj)==1
0131             ret=subsref(obj, index(2:end));
0132         else
0133             error('Single element of ''%s'' must be specified', inputname(1));
0134         end
0135     end
0136 end
0137 return
0138 end
0139
0140 %-----
0141 function CheckErr()
0142 % CheckErr checks that the exception has been thrown for the expected
0143 % reason - if not rethrow it.
0144 err=lasterror();
0145 if strcmp(err.identifier, 'MATLAB:unassignedOutputs');
0146     return
0147 else
0148     rethrow(err);
0149 end
0150 return
0151 end
0152 %-----

```

Contents

PURPOSE ^

%% path: m/gui/@jcontrol/private
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%% path: m/gui/@jcontrol/private %%%
 <a href="matlab:help
m/gui/@jcontrol/private/VisibleProperty">m/gui/@jcontrol/private/VisibleProperty - -
 helper function for JCONTROL methods

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

VisibleProperty

PURPOSE ^

VisibleProperty - helper function for JCONTROL methods

SYNOPSIS ^

function val=VisibleProperty(val)

DESCRIPTION ^

```
VisibleProperty - helper function for JCONTROL methods
Converts numeric inputs to string for visibility property.
Example:
val=VisibleProperty(val)

-----
Author: Malcolm Lidierth 07/07
Copyright The Author & King's College London 2007
-----
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function val=VisibleProperty(val)
0002 % VisibleProperty - helper function for JCONTROL methods
0003 %
0004 % Converts numeric inputs to string for visibility property.
0005 %
0006 % Example:
0007 % val=VisibleProperty(val)
0008 %
0009 % -----
0010 % Author: Malcolm Lidierth 07/07
0011 % Copyright The Author & King's College London 2007
0012 % -----
0013
0014 switch val
0015     case 0
0016         val='off';
0017     case 1
0018         val='on';
0019     %otherwise
0020         % return unchanged
0021 end
0022 return
0023 end
```


Contents

PURPOSE [^]

%%%%%%%%%%%% path: m/gui/ao_browser
%%%%%%%%%

SYNOPSIS [^]

This is a script file.

DESCRIPTION [^]

```
%%%%%%%%%%%% path: m/gui/ao_browser %%%%%%%%%%  
    <a href="matlab:help m/gui/ao_browser/ltpda_explorer">m/gui/ao_browser/ltpda_explorer</a>  
-   for exploring analysis objects and plotting/displaying their fields  
    <a href="matlab:help  
m/gui/ao_browser/ltpda_javacomponent">m/gui/ao_browser/ltpda_javacomponent</a> - WARNING: This  
feature is not supported in MATLAB  
    <a href="matlab:help m/gui/ao_browser/ltpda_uitree">m/gui/ao_browser/ltpda_uitree</a>  
-   WARNING: This feature is not supported in MATLAB  
    <a href="matlab:help  
m/gui/ao_browser/ltpda_uitreenode">m/gui/ao_browser/ltpda_uitreenode</a>      - WARNING: This  
feature is not supported in MATLAB
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

ltpda_explorer

PURPOSE ^

LTPDA_EXPLORER for exploring analysis objects and plotting/displaying their fields

SYNOPSIS ^

function ltpda_explorer(varargin)

DESCRIPTION ^

```
LTPDA_EXPLORER for exploring analysis objects and plotting/displaying their fields
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_EXPLORER for exploring analysis objects and plotting/displaying
              their fields. It is possible to browse through the history tree
              to see only a part of this tree. If the user will use the plot
              function the plot will be printed in a new figure.

CALL:      ltpda_explorer;                % Read the ao's from the 'base' workspace
           ltpda_explorer(ao);
           ltpda_explorer(ao_vector);
           ltpda_explorer(ao_matrix);

VERSION: $Id: ltpda_explorer.m,v 1.2 2008/08/04 12:54:29 hewitson Exp $

HISTORY: 10-06-07 Diepholz
           Creation

NOTE:     The idea and the core source code are taken from:
           Hassan Lahdili (hassan.lahdili@crc.ca)
           Communications Research Centre (CRC) | Advanced Audio Systems (AAS)
           www.crc.ca | www.crc.ca/aas
           Ottawa, Canada
           CRC Advanced Audio Systems - Ottawa 16/02/2005 2004-2005
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpda_uitree](#) WARNING: This feature is not supported in MATLAB
-  [ltpda_uitreenode](#) WARNING: This feature is not supported in MATLAB

This function is called by:

SUBFUNCTIONS ^

-  function [mouse_cb\(h, ev\)](#)
-  function [f_disp_menu1\(h, ev\)](#)
-  function [f_tree_menu1\(h, ev\)](#)
-  function [f_tree_menu2\(h, ev\)](#)
-  function [plotselected_cb\(h, ev\)](#)
-  function [displayselected_cb\(h, ev\)](#)
-  function [cNode = nodeSelected_cb4\(tree, ev\)](#)
-  function [nodes = myExpfcn4\(tree, value\)](#)
-  function [cNode = nodeWillExpand_cb4\(tree, ev\)](#)

 function [val, displayed, cNode] = getNodevalue(cNode, s)
 function c_str = split_by_comma(str, pref)

SOURCE CODE [^]

```

0001 function ltpda_explorer(varargin)
0002 % LTPDA_EXPLORER for exploring analysis objects and plotting/displaying their fields
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDA_EXPLORER for exploring analysis objects and plotting/displaying
0007 %               their fields. It is possible to browse through the history tree
0008 %               to see only a part of this tree. If the user will use the plot
0009 %               function the plot will be printed in a new figure.
0010 %
0011 % CALL:        ltpda_explorer;                % Read the ao's from the 'base' workspace
0012 %               ltpda_explorer(ao);
0013 %               ltpda_explorer(ao_vector);
0014 %               ltpda_explorer(ao_matrix);
0015 %
0016 %
0017 % VERSION: $Id: ltpda_explorer.m,v 1.2 2008/08/04 12:54:29 hewitson Exp $
0018 %
0019 % HISTORY: 10-06-07 Diepholz
0020 %           Creation
0021 %
0022 % NOTE:       The idea and the core source code are taken from:
0023 %               Hassan Lahdili (hassan.lahdili@crc.ca)
0024 %               Communications Research Centre (CRC) | Advanced Audio Systems (AAS)
0025 %               www.crc.ca | www.crc.ca/aas
0026 %               Ottawa, Canada
0027 %               CRC Advanced Audio Systems - Ottawa 16/02/2005 2004-2005
0028 %
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030 %
0031 %
0032 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0033 %               Define the Positions %
0034 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0035 %
0036 % figure position
0037 % FIG_X   = .150;
0038 % FIG_Y   = .150;
0039 % FIG_dX  = .700;
0040 % FIG_dY  = .600;
0041 %
0042 FIG_X    = 150;
0043 FIG_Y    = 150;
0044 FIG_dX   = 700;
0045 FIG_dY   = 600;
0046 %
0047 %
0048 % tree position
0049 TREE_X   = .000;
0050 TREE_Y   = .000;
0051 TREE_dX  = .350;
0052 TREE_dY  = 1.000;
0053 %
0054 % plot field position
0055 PLOT_X   = .370;
0056 PLOT_Y   = .105;
0057 PLOT_dX  = .610;
0058 PLOT_dY  = .720;
0059 %
0060 % display field position
0061 DISP_X   = TREE_X;
0062 DISP_Y   = .750;
0063 DISP_dX  = TREE_dX;
0064 DISP_dY  = 1-DISP_Y;
0065 %
0066 % explorer name position
0067 EXPL_NAME_dX = .370;
0068 EXPL_NAME_dY = .045;
0069 EXPL_NAME_X  = PLOT_X+(PLOT_dX-EXPL_NAME_dX)/2;
0070 EXPL_NAME_Y  = .030;
0071 % EXPL_NAME_Y = .065;
0072 %
0073 % info fields position
0074 N_INFOS = 3;
0075 %
0076 INFO_X   = PLOT_X;
0077 INFO_Y   = .915;

```

```

0078 INFO_dX = PLOT_dX / N_INFOS;
0079 INFO_dY = .034;
0080
0081 BG_COLOR = [.925 .914 .847];
0082
0083 MAX_HIST_DEPTH = 4;
0084
0085 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0086 % Check the input %
0087 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0088
0089
0090 % Get AO list from workspace
0091 vars = evalin('base','who');
0092 aoList = [];
0093 for j=1:length(vars)
0094     var = evalin('base', vars{j});
0095     if isa(var, 'ao')
0096         % add to list
0097         aoList = [aoList var];
0098     end
0099 end
0100
0101 obj      = [];
0102 obj_value = {};
0103 obj_name  = {};
0104
0105 % Read the ao's from the 'base' workspace
0106
0107 if nargin == 0
0108     tree_name = 'obj_ws';
0109     ws_vars   = evalin('base','whos');
0110 elseif nargin == 1
0111     tree_name = inputname(1);
0112     tmp       = varargin{1};
0113
0114     if iscell(tmp)
0115         ws_vars = [];
0116         for jj = 1:length(tmp)
0117             ws_vars(jj).name = [inputname(1) '{' num2str(jj) '}'];
0118             ws_vars(jj).class = class(tmp{jj});
0119             ws_vars(jj).obj   = tmp{jj};
0120         end
0121     else
0122         ws_vars = whos('tmp');
0123         ws_vars.name = inputname(1);
0124         ws_vars.obj = tmp;
0125     end
0126 else
0127     error('#####');
0128 end
0129
0130 for ii=1:length(ws_vars)
0131     if nargin == 0
0132         obj = evalin('base', ws_vars(ii).name);
0133     elseif nargin == 1
0134         obj = ws_vars(ii).obj;
0135     else
0136         error('#####');
0137     end
0138
0139     if utils.helper.isobject(obj)
0140
0141         % the object in the workspace is a single value
0142         if numel(obj) == 1
0143             obj_value{end+1} = obj;
0144             obj_name{end+1} = [ws_vars(ii).class ':' ws_vars(ii).name];
0145         else
0146             [n,m] = size(obj);
0147
0148             % the ao in the workspace is a vector
0149             if n == 1 || m == 1
0150                 for jj=1:length(obj)
0151                     obj_value{end+1} = obj(jj);
0152                     obj_name{end+1} = [ws_vars(ii).class ':' ws_vars(ii).name '(' num2str(jj)
0153 ')'];
0154                 end
0155
0156                 % the ao in the workspace is a matrix
0157                 elseif n > 1 && m > 1

```

```

0163
0164     for gg = 1:n
0165         for hh = 1:m
0166             obj_value{end+1} = obj(gg, hh);
0167             obj_name{end+1} = [ws_vars(ii).class ':' ws_vars(ii).name '(' num2str(gg)
0168             num2str(hh) ')'];
0169         end
0170     end
0171 else
0172     error('### this should not happen. ');
0173 end
0174 end
0175 end
0176 end
0177 end
0178 end
0179
0180 fig_name = ['explore the object: ' tree_name];
0181
0182 % 'Units',          'normalized',      ...
0183
0184 % Define figure
0185 fig = figure('NextPlot', 'add', ...
0186             'NumberTitle', 'off', ...
0187             'ToolBar', 'none', ...
0188             'name', fig_name, ...
0189             'Color', BG_COLOR, ...
0190             'ToolBar', 'none', ...
0191             'NextPlot', 'new', ...
0192             'MenuBar', 'none', ...
0193             'Position', [FIG_X FIG_Y
0194             FIG_dX FIG_dY]);
0195
0196 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0197 % Define the tree %
0198 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0199
0200 root = ltpda_uitreenode(tree_name, tree_name, [], false);
0201 tree = ltpda_uitree( fig, 'Root', root, ...
0202                    'ExpandFcn', @myExpFcn4);
0203
0204 set(tree, 'Units', 'normalized', ...
0205         'position', [TREE_X TREE_Y
0206                     TREE_dX TREE_dY], ...
0207         'NodeWillExpandCallback', @nodeWillExpand_cb4, ...
0208         'NodeSelectedCallback', @nodeSelected_cb4);
0209 tmp = tree.FigureComponent;
0210 cell_Data = cell(3,1);
0211
0212 % cell_Data{1} = varargin{:};
0213 cell_Data{1} = obj_value;
0214 cell_Data{3} = obj_name;
0215
0216 set(tmp, 'UserData', cell_Data);
0217
0218 % Define the plot field
0219 haxes = axes('Units', 'normalized', ...
0220             'Position', [PLOT_X PLOT_Y
0221                         PLOT_dX PLOT_dY], ...
0222             'Box', 'on', ...
0223             'XTick', [], ...
0224             'YTick', []);
0225
0226 box off;
0227 axis off;
0228
0229 % Define the info fields 'name'
0230 txt1 = uicontrol('String', '', ...
0231                 'Units', 'normalized', ...
0232                 'Style', 'Edit', ...
0233                 'Position', [INFO_X INFO_Y
0234                             INFO_dX INFO_dY], ...
0235                 'BackgroundColor', BG_COLOR);
0236
0237 % Define the info fields 'size'
0238 txt2 = uicontrol('String', '', ...
0239                 'Units', 'normalized', ...
0240                 'Style', 'Edit', ...
0241                 'Position', [(INFO_X+INFO_dX) INFO_Y
0242                             INFO_dX INFO_dY], ...
0243                 'BackgroundColor', BG_COLOR);
0244
0245 % Define the info fields 'class'
0246 txt3 = uicontrol('String', '', ...
0247                 'Units', 'normalized', ...
0248                 'Style', 'Edit', ...

```

```

0248         'Position',          [(INFO_X+2*INFO_dX) INFO_Y ...
0249                             INFO_dX          INFO_dY], ...
0250         'BackgroundColor',    BG_COLOR);
0251 % Define the info fields 'value'
0252 txt4 = uicontrol('String',    '', ...
0253                 'Units',      'normalized', ...
0254                 'Style',      'Edit', ...
0255                 'Position',    [ INFO_X (INFO_Y-INFO_dY) ...
0256                               3*INFO_dX INFO_dY], ...
0257                 'BackgroundColor', BG_COLOR);
0258
0259 % Define the info fields 'value'
0260 txt5 = uicontrol('String',    '', ...
0261                 'Units',      'normalized', ...
0262                 'Style',      'listbox', ...
0263                 'Visible',    'off', ...
0264                 'FontSize',    8, ...
0265                 'Position',    [DISP_X DISP_Y ...
0266                               DISP_dX DISP_dY], ...
0267                 'BackgroundColor', BG_COLOR);
0268
0269
0270
0271 % Define the decription of the info fields
0272 col1 = uicontrol('String',    'Name', ...
0273                 'Units',      'normalized', ...
0274                 'Style',      'Text', ...
0275                 'Position',    [INFO_X (INFO_Y+INFO_dY) ...
0276                               INFO_dX INFO_dY], ...
0277                 'BackgroundColor', BG_COLOR);
0278 col2 = uicontrol('String',    'Size', ...
0279                 'Units',      'normalized', ...
0280                 'Style',      'Text', ...
0281                 'Position',    [(INFO_X+INFO_dX) (INFO_Y+INFO_dY) ...
0282                               INFO_dX INFO_dY], ...
0283                 'BackgroundColor', BG_COLOR);
0284 col3 = uicontrol('String',    'Class', ...
0285                 'Units',      'normalized', ...
0286                 'Style',      'Text', ...
0287                 'Position',    [(INFO_X+2*INFO_dX) (INFO_Y+INFO_dY) ...
0288                               INFO_dX INFO_dY], ...
0289                 'BackgroundColor', BG_COLOR);
0290
0291 % Define the name of the explorer
0292 expl_name = uicontrol('String',    'Analysis Object explorer', ...
0293                      'Units',      'normalized', ...
0294                      'Style',      'text', ...
0295                      'Position',    [EXPL_NAME_X EXPL_NAME_Y ...
0296                                      EXPL_NAME_dX EXPL_NAME_dY], ...
0297                      'ForegroundColor', [0.2 0.4 1], ...
0298                      'FontSize',    18, ...
0299                      'FontWeight',  'bold', ...
0300                      'FontAngle',   'italic');
0301
0302 tree_menu = uicontextmenu();
0303 tree_menu1 = uimenu(tree_menu, 'Label', 'Plot', ...
0304                             'Callback', @f_tree_menu1);
0305 tree_menu2 = uimenu(tree_menu, 'Label', 'Display', ...
0306                             'Callback', @f_tree_menu2);
0307
0308 disp_menu = uicontextmenu();
0309 disp_menu1 = uimenu(disp_menu, 'Label', 'close', ...
0310                        'Callback', @f_disp_menu1);
0311
0312 set(tree.Tree, 'MousePressedCallback', @mouse_cb);
0313 set(tree.Tree, 'UIContextMenu', tree_menu);
0314
0315 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0316 % Mouse Pressed Handler
0317 function mouse_cb(h, ev)
0318     if ev.getModifiers()== ev.META_MASK
0319         % Workaround to set the y position
0320         % Workaround to set the x position
0321         vis = get(get(ev, 'Component'), 'VisibleRect');
0322         x_width = get(get(ev, 'Component'), 'Width');
0323         y_height = get(get(ev, 'Component'), 'Height');
0324
0325         new_x = ev.getX-vis(1);
0326         new_y = -ev.getY+y_height-(y_height-vis(4));
0327         set(tree_menu, 'Position', [new_x new_y], ...
0328             'Visible', 'on');
0329     end
0330
0331 end
0332 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0333 function f_disp_menu1(h, ev)

```

```

0334     set(txt5, 'Visible', 'off')
0335     set(tree, 'position', [TREE_X TREE_Y ...
0336                          TREE_dx TREE_dy])
0337 end
0338 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0339 function f_tree_menu1(h,ev)
0340     plotselected_cb;
0341 end
0342 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0343 function f_tree_menu2(h,ev)
0344     displayselected_cb;
0345 end
0346 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0347 function plotselected_cb(h, ev)
0348
0349     tmp = tree.FigureComponent;
0350     S = get(tmp, 'UserData');
0351     s = S{1};
0352     cNode = S{2};
0353     [val, plotted, cNode] = getcNodevalue(cNode, s);
0354
0355     cla(haxes)
0356
0357     %%%% Plot history object %%%%
0358     if (isa(val,'history'))
0359         if length(val) == 1
0360             figure;
0361             plot (val);
0362             ii = strfind(plotted, 'inhists');
0363             title(haxes, sprintf('History-Level: %d', length(ii)+1))
0364         else
0365             na = text(0.5,0.5,'Select the left or right branch. ');
0366             set(na, 'HorizontalAlignment', 'center', ...
0367                 'Color', 'r', ...
0368                 'FontWeight', 'bold', ...
0369                 'EdgeColor', 'k', ...
0370                 'BackgroundColor', 'w', ...
0371                 'FontSize', 10, ...
0372                 'Margin', 5);
0373         end
0374
0375     %%%% Plot data object %%%%
0376     elseif isa(val,'fsdata') || isa(val,'tsdata') || ...
0377            isa(val,'xydata') || isa(val,'cdata')
0378         figure;
0379         plot(ao(val))
0380
0381     %%%% Plot the A0 object %%%%
0382     elseif isa(val, 'ao')
0383         figure;
0384         plot(ao(val))
0385
0386     %%%% Plot mfir and miir object %%%%
0387     elseif isa(val, 'mfir') || isa(val, 'miir')
0388         figure;
0389         resp(val)
0390
0391     %%%% Plot pzmodel object %%%%
0392     elseif isa(val, 'pzmodel')
0393         resp(val)
0394
0395     %%%% Is the parent node == 'data' so plot data %%%%
0396     else
0397
0398         cNode = S{2};
0399         parent = '';
0400
0401         if cNode.getLevel >= 1
0402             parent = cNode.getParent;
0403         end
0404
0405         if strcmp(parent.getValue, 'data')
0406             grandpa = parent.getParent;
0407             grandpa_val = grandpa.getValue;
0408             von = findstr(grandpa_val, '(');
0409             bis = findstr(grandpa_val, ')');
0410             if ~isempty(von) && ~isempty(bis)
0411                 index = grandpa_val(von+1:bis-1);
0412             else
0413                 index = '1';
0414             end
0415
0416             eval (sprintf('val = S{1}(%s);',index))
0417             figure;
0418             plot(val);
0419         end

```



```

0420
0421     end
0422
0423 end
0424 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0425 function displayselected_cb(h, ev)
0426     tmp = tree.FigureComponent;
0427     S = get(tmp, 'UserData');
0428     s = S{1};
0429     cNode = S{2};
0430     [val, displayed, cNode] = getcNodevalue(cNode, s);
0431
0432     text1 = '';
0433     if isobject(val)
0434         text1 = display(val);
0435     else
0436         disp(val)
0437     end
0438
0439     % some sisplay outputs contains '\n' <-> char(10)
0440     % text can not display this character so replace it with ' '
0441     text1 = strrep(text1, char(10), ' ');
0442
0443     set(tree, 'position', [TREE_X TREE_Y ...
0444         TREE_dX TREE_dY-DISP_dY]);
0445     set(txt5, 'string', text1);
0446     set(txt5, 'Visible', 'on');
0447     set(txt5, 'UIContextMenu', disp_menu);
0448 end
0449 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0450
0451 function cNode = nodeSelected_cb4(tree, ev)
0452     cNode = ev.getCurrentNode;
0453     tmp = tree.FigureComponent;
0454     cell_Data = get(tmp, 'UserData');
0455     cell_Data{2} = cNode;
0456     s = cell_Data{1};
0457     val = s;
0458     plotted = cNode.getValue;
0459     selected = plotted;
0460     [val, plotted, cNode] = getcNodevalue(cNode, val);
0461     set(txt1, 'string', selected)
0462     set(txt2, 'string', strcat(num2str(size(val,1)), 'x', num2str(size(val,2))) )
0463     set(txt3, 'string', class(val))
0464
0465     str = ' ';
0466     c_str = {};
0467     cla(haxes)
0468
0469     if ~isempty(val)
0470
0471         if isnumeric(val)
0472             %normalize the vector
0473             si = size(val);
0474             if si(1) > si(2)
0475                 val = val.';
0476             else
0477                 val = val;
0478             end
0479             if size(val,1) == 1 || size(val,2) == 1
0480                 if length(val) > 3
0481                     str = strcat(num2str(val(1:3)), ' ...');
0482                 else
0483                     str = num2str(val);
0484                 end
0485             else
0486                 str = 'Matrix';
0487             end
0488
0489             elseif ischar(val)
0490                 str = val;
0491
0492             elseif islogical(val)
0493                 if val
0494                     str = 'true';
0495                 else
0496                     str = 'false';
0497                 end
0498
0499             elseif isobject(val)
0500                 if isa(val, 'ao')
0501                     str = 'Analysis Object';
0502                 elseif isa(val, 'cdata')
0503                     str = 'C-Data Object';
0504                 elseif isa(val, 'fsdata')
0505                     str = 'Frequency-Series Object';

```



```

0506 elseif isa(val, 'tsdata')
0507     str = 'Time-Series Object';
0508 elseif isa(val, 'xydata')
0509     str = 'X-Y Data Object';
0510
0511 elseif isa(val, 'history')
0512     str = 'History Object';
0513     if length(val) == 1
0514         pl = plist(param('max_depth', MAX_HIST_DEPTH));
0515         plot(haxes, val, pl);
0516         ii = strfind(plotted, 'inhists');
0517         title(haxes, sprintf('History-Level: %d', length(ii)+1))
0518     else
0519         c_str{1} = 'Select the left or right branch.';
0520     end
0521
0522 elseif isa(val, 'param')
0523     if length(val) == 1
0524         str = char(val);
0525         c_str = split_by_comma(str);
0526     else
0527         for ii = 1:length(val)
0528             str = char(val(ii));
0529             c_str1 = split_by_comma(str, '- ');
0530             c_str(end+1:end+length(c_str1)) = c_str1;
0531         end
0532     end
0533     str = 'Parameter Object';
0534
0535 elseif isa(val, 'plist')
0536     if numel(val) > 1
0537         c_str{1} = 'select a plist';
0538     else
0539         for ii=1:length(val.params)
0540             str = char(val.params(ii));
0541             c_str1 = split_by_comma(str, '- ');
0542             c_str(end+1:end+length(c_str1)) = strrep(c_str1, '_', '\_');
0543         end
0544     end
0545     str = 'Parameter List Object';
0546
0547 elseif isa(val, 'mfir')
0548     str = 'FIR Filter Object';
0549 elseif isa(val, 'miir')
0550     str = 'IIR Filter Object';
0551 elseif isa(val, 'pole')
0552     str = 'Pole Object';
0553 elseif isa(val, 'provenance')
0554     str = 'Provenance Object';
0555 elseif isa(val, 'pzmodel')
0556     str = 'Pole Zero Object';
0557 elseif isa(val, 'specwin')
0558     str = 'Spectral Window Object';
0559 elseif isa(val, 'zero')
0560     str = 'Zero Object';
0561 elseif isa(val, 'time')
0562     str = 'Time Object';
0563 else
0564     str = 'Unknown class';
0565 end
0566
0567 elseif iscell(val)
0568     for i = 1:min(length(val),3)
0569         if ischar(val{i})
0570             str = strcat(str, val{i});
0571         elseif isnumeric(val)
0572             str = strcat(str, num2str(val{i}));
0573         end
0574         if i < min(length(val),3)
0575             str = strcat(str, ',');
0576         end
0577     end
0578     if length(val) > 3
0579         str = strcat(str, '...');
0580     end
0581
0582 end
0583
0584 if ~isempty(c_str)
0585     c_str = strtrim(c_str);
0586     na = text(0.5,0.5,c_str);
0587     txt_extent = get(na, 'Extent');
0588     set(na, 'Position', [0.5-txt_extent(3)/2, 0.5], ...
0589         'HorizontalAlignment', 'left', ...
0590         'Color', 'k', ...

```

```

0592         'FontWeight',          'bold', ...
0593         'EdgeColor',           'k', ...
0594         'BackgroundColor',     'w', ...
0595         'FontSize',            10, ...
0596         'Margin',              5);
0597     end
0598
0599     else % ~isempty(val)
0600         str = 'The field is empty';
0601     end
0602     set(txt4, 'string', str)
0603     set(tmp, 'UserData', cell_Data);
0604 end
0605
0606 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0607 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0608 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0609
0610 end
0611
0612 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0613 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0614 function nodes = myExpfcn4(tree,value)
0615
0616     tmp = tree.FigureComponent;
0617     S = get(tmp, 'UserData');
0618     s = S{1};
0619     cNode = S{2};
0620     [val, cNode] = getNodevalue(cNode, s);
0621
0622     % Set the path to the *.gif files
0623     % This tis the current path of this function + 'exp_struct_icons'
0624     pth = '';
0625     eval(sprintf('pth = which('%s');',mfilename))
0626     index = find(pth==filesep, 1, 'last');
0627     pth = pth(1:index);
0628     pth = [pth 'exp_struct_icons' filesep];
0629
0630     [n,m] = size(val);
0631     count = 0;
0632
0633     %% Vector or Matrix
0634     if m>1 || n>1
0635
0636         if isa(val, 'ao')
0637             iconpath = [pth, 'analysis_object.gif'];
0638         else
0639             iconpath = [pth, 'struct_icon.gif'];
0640         end
0641
0642         %% Vector
0643         if m==1 || n==1
0644             L = length(val);
0645
0646             for J = 1:L
0647                 count = count + 1;
0648                 cNode = S{2};
0649
0650                 level = cNode.getLevel;
0651                 fname = strcat(cNode.getValue, '(' , num2str(J), ')');
0652
0653                 if level==0 && ~isempty(S{3}) && numel(S{3}) == numel(S{1})
0654                     node_str = S{3}(J);
0655                 else
0656                     node_str = fname;
0657                 end
0658
0659                 nodes(count) = ltpda_uitreenode(fname, node_str, iconpath, 0);
0660             end
0661         %% Matrix
0662         else
0663             for ii=1:n
0664                 for jj=1:m
0665                     count = count + 1;
0666                     cNode = S{2};
0667                     fname = [cNode.getValue '(' num2str(ii) ',' num2str(jj) ')'];
0668                     nodes(count) = ltpda_uitreenode(fname, fname, iconpath, 0);
0669                 end
0670             end
0671         end
0672
0673     end
0674     %% Struct, Object or single value
0675     else
0676         %%
0677         val = val;

```

```

0678         fnames = fieldnames(val);
0679
0680     for i=1:length(fnames)
0681         count = count + 1;
0682         x = getfield(val, fnames{i});
0683
0684         if isa(x, 'ao')
0685             iconpath = [pth, 'analysis_object.gif'];
0686         elseif isa(x, 'tsdata')
0687             iconpath = [pth, 'ts_data.gif'];
0688         elseif isa(x, 'fsdata')
0689             iconpath = [pth, 'fs_data.gif'];
0690         elseif isa(x, 'xydata')
0691             iconpath = [pth, 'xy_data.gif'];
0692         elseif isa(x, 'cdata')
0693             iconpath = [pth, 'c_data.gif'];
0694         elseif isa(x, 'history')
0695             iconpath = [pth, 'history.gif'];
0696         elseif isa(x, 'plist')
0697             iconpath = [pth, 'plist.gif'];
0698
0699         elseif isstruct(x)
0700             if length(x) > 1
0701                 iconpath = [pth, 'structarray_icon.gif'];
0702             else
0703                 iconpath = [pth, 'struct_icon.gif'];
0704             end
0705         elseif isnumeric(x)
0706             iconpath = [pth, 'double_icon.gif'];
0707         elseif iscell(x)
0708             iconpath = [pth, 'cell_icon.gif'];
0709         elseif ischar(x)
0710             iconpath = [pth, 'char_icon.gif'];
0711         elseif islogical(x)
0712             iconpath = [pth, 'logic_icon.gif'];
0713         elseif isobject(x)
0714             iconpath = [pth, 'obj_icon.gif'];
0715         else
0716             iconpath = [pth, 'unknown_icon.gif'];
0717         end
0718
0719         if isstruct(x) || isobject(x)
0720             isLeaf = 0;
0721         else
0722             isLeaf = 1;
0723         end
0724
0725         nodes(count) = ltpda_uitreenode(fnames{i}, fnames{i}, iconpath, isLeaf);
0726     end
0727 end
0728
0729 if (count == 0)
0730     nodes = [];
0731 end
0732 end
0733
0734 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0735
0736 function cNode = nodeWillExpand_cb4(tree, ev)
0737     cNode = ev.getCurrentNode;
0738     tmp = tree.FigureComponent;
0739     cell_Data = get(tmp, 'UserData');
0740     cell_Data{2} = cNode;
0741     set(tmp, 'UserData', cell_Data);
0742 end
0743
0744 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0745 function [val, displayed, cNode] = getNodevalue(cNode, s)
0746
0747     fields = {};
0748     while cNode.getLevel ~= 0
0749         fields = [fields; cNode.getValue];
0750         c = findstr(cNode.getValue, '(');
0751         if ~isempty(c) && cNode.getLevel ~= 0
0752             cNode = cNode.getParent;
0753         end
0754
0755         if cNode.getLevel == 0, break; end
0756         cNode = cNode.getParent;
0757     end
0758
0759     val = s;
0760
0761     if ~isempty(fields)
0762         L=length(fields);
0763         displayed = fields{L};

```

```

0764 % create the variable: displayed
0765 for j = L-1:-1:1
0766     displayed = strcat(displayed, '.', fields{j});
0767 end
0768
0769 for i = L:-1:1
0770     field = fields{i};
0771     von = findstr(field, '(');
0772     bis = findstr(field, ')');
0773     if ~isempty(von)
0774
0775         idx = field(von+1:bis-1);
0776         field = field(1:von-1);
0777         if (strcmp(field, cNode.getValue))
0778             cmd = sprintf('val = val(%s);', idx);
0779             eval(cmd);
0780             if iscell(val) && numel(val) == 1
0781                 val = val{1};
0782             else
0783                 error('##### MAch mich neu');
0784             end
0785         else
0786
0787             cmd = sprintf('val = getfield(val, field, {%s});', idx);
0788             eval(cmd);
0789             if iscell(val) && numel(val) == 1
0790                 val = val{1};
0791             end
0792         end
0793     end
0794
0795     else
0796         if iscell(val) && numel(val) == 1
0797             val = val{1};
0798         elseif numel(val) ~= 1
0799             error('##### MAch mich neu');
0800         end
0801         val = getfield(val, field);
0802     end
0803 end
0804 else
0805     displayed = cNode.getValue;
0806     if iscell(val) && numel(val) == 1
0807         val = val{1};
0808     else
0809         end
0810     return;
0811 end
0812 end
0813
0814
0815 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0816 function c_str = split_by_comma(str, pref)
0817
0818     if nargin < 2
0819         pref = ',';
0820     end
0821
0822     c_str = {};
0823     c_str{1} = str;
0824     index = find(str==',' );
0825     von = 1;
0826     for ii = 1:length(index)
0827         bis = index(ii)-1;
0828         c_str{ii} = [pref str(von:bis)];
0829         von = bis + 2;
0830     end
0831     if ~isempty(index)
0832         c_str{ii+1} = [pref str(von:end)];
0833     end
0834
0835     c_str = strtrim(c_str);
0836 end
0837

```

[Home](#) > [m](#) > [gui](#) > [ao_browser](#) > ltpda_javacomponent.m

ltpda_javacomponent

PURPOSE [^]

WARNING: This feature is not supported in MATLAB

SYNOPSIS [^]

function [hcomponent, hcontainer] = ltpda_javacomponent(component, position, parent)

DESCRIPTION [^]

WARNING: This feature is not supported in MATLAB and the API and functionality may change in a future release.

JAVACOMPONENT Create a Java AWT Component and put it in a figure

JAVACOMPONENT(COMPONENTNAME) creates the Java component specified by the string COMPONENTNAME and places it in the current figure or creates a new figure if one is not available. The default position is [20 20 60 20].
NOTE: This is a thread safe way to create and embed a java component in a figure window. For more thread safe functions to create and modify java components, see AWTCREATE, AWTINVOKE.

JAVACOMPONENT(HCOMPONENT) places the Java component HCOMPONENT in the current figure or creates a new figure if one is not available. The default position is [20 20 60 20].

JAVACOMPONENT(..., POSITION, PARENT) places the Java component in the specified PARENT at position POSITION. PARENT can be a Figure or a Uipanel. POSITION is in pixel units with the format [left, bottom, width, height].

JAVACOMPONENT(..., CONSTRAINT, PARENT) places the Java component next to the figure's drawing area using CONSTRAINT. CONSTRAINT that can be NORTH, SOUTH, EAST, OR WEST placement - following Java AWT's BorderLayout rules. The handle to the Java component is returned on success, empty is returned on error. If the parent is a uipanel, the component is placed in the parent figure of the uipanel. If parent is a uitoolbar handle, CONSTRAINT is ignored and the component is placed last in the child list for the given toolbar.

[HCOMPONENT, HCONTAINER] = JAVACOMPONENT(...) returns the handle to the Java component in HCOMPONENT and its HG container in HCONTAINER. HCONTAINER is only returned when pixel positioning is used. It should be used to change the units, position, and visibility of the Java component after it is added to the figure.

Examples:

```
f = figure;
b = javacomponent('javax.swing.JButton'); % Thread safe creation
set(b,'ActionPerformedCallback','disp Hi!');

f = figure('WindowState', 'docked');
b1 = awtcreate('javax.swing.JButton', 'Ljava.lang.String;', 'Hi!');
set(b1,'ActionPerformedCallback','disp Hi!');
javacomponent(b1);

f = figure;
comp = awtcreate('javax.swing.JSpinner');
[comp, container] = javacomponent(comp);
set(container,'Position', [100, 100, 100, 40]);
set(container,'Units', 'normalized');

f = figure;
p = uipanel('Position', [0 0 .2 1]);
ppos = getpixelposition(p);
tr = awtcreate('javax.swing.JTree');
[tree treecontainer] = javacomponent(tr, [0 0 ppos(3) ppos(4)], p);

f = figure('WindowState', 'docked');
ta = awtcreate('javax.swing.JTable', 'II', 3, 10);
```

```

table = javacomponent(ta, java.awt.BorderLayout.SOUTH, f);
f = figure;
tb = uitoolbar(f);
b2 = awtcreate('javax.swing.JButton', 'Ljava.lang.String;', 'Hi again!');
b2 = javacomponent(b2, [], tb); % Note: Position is ignored.
set(b2, 'ActionPerformedCallback', 'disp(''Hi again!'')');

```

See also [USEJAVACOMPONENT](#), [AWTCREATE](#), [AWTINVOKE](#)

Copyright 1984-2006 The MathWorks, Inc.
 \$Revision: 1.2 \$ \$Date: 2008/01/22 20:45:22 \$

Adapted for LTPDA

\$Id: ltpda_javacomponent.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

-  [function createPanel](#)
-  [function handleResize\(obj, evd\)](#)
-  [function handleVisible\(obj, evd\)](#)
-  [function handlePreParent\(obj, evd\)](#)
-  [function handlePostParent\(obj, evd\)](#)
-  [function configureComponent](#)
-  [function containerDelete\(obj, evd, hc\)](#)
-  [function componentDelete\(obj, evd, hUicontainer, parentIsFigure\)](#)
-  [function str=usage](#)

SOURCE CODE [^]

```

0001 function [hcomponent, hcontainer] = ltpda_javacomponent(component, position, parent)
0002 % WARNING: This feature is not supported in MATLAB
0003 % and the API and functionality may change in a future release.
0004 %
0005 % JAVACOMPONENT Create a Java AWT Component and put it in a figure
0006 %
0007 % JAVACOMPONENT(COMPONENTNAME) creates the Java component specified by the
0008 % string COMPONENTNAME and places it in the current figure or creates a new
0009 % figure if one is not available. The default position is [20 20 60 20].
0010 % NOTE: This is a thread safe way to create and embed a java component in a
0011 % figure window. For more thread safe functions to create and modify java
0012 % components, see AWTCREATE, AWTINVOKE.
0013 %
0014 % JAVACOMPONENT(HCOMPONENT) places the Java component HCOMPONENT in the
0015 % current figure or creates a new figure if one is not available. The
0016 % default position is [20 20 60 20].
0017 %
0018 % JAVACOMPONENT(..., POSITION, PARENT) places the Java component in the
0019 % specified PARENT at position POSITION. PARENT can be a Figure or a
0020 % Uipanel. POSITION is in pixel units with the format [left, bottom, width,
0021 % height].
0022 %
0023 % JAVACOMPONENT(..., CONSTRAINT, PARENT) places the Java component next to
0024 % the figure's drawing area using CONSTRAINT. CONSTRAINT that can be NORTH,
0025 % SOUTH, EAST, OR WEST placement - following Java AWT's BorderLayout rules.
0026 % The handle to the Java component is returned on success, empty is
0027 % returned on error. If the parent is a uipanel, the component is placed in
0028 % the parent figure of the uipanel. If parent is a uitoolbar handle,
0029 % CONSTRAINT is ignored and the component is placed last in the child list
0030 % for the given toolbar.
0031 %
0032 % [HCOMPONENT, HCONTAINER] = JAVACOMPONENT(...)
0033 % returns the handle to the Java component in HCOMPONENT and its HG
0034 % container in HCONTAINER. HCONTAINER is only returned when pixel
0035 % positioning is used. It should be used to change the units, position,

```



```

0036 % and visibility of the Java component after it is added to the figure.
0037 %
0038 %
0039 %     Examples:
0040 %
0041 %     f = figure;
0042 %     b = javacomponent('javax.swing.JButton'); % Thread safe creation
0043 %     set(b,'ActionPerformedCallback','disp Hi!');
0044 %
0045 %     f = figure('WindowState','docked');
0046 %     b1 = awtcreate('javax.swing.JButton','Ljava.lang.String;', 'Hi!');
0047 %     set(b1,'ActionPerformedCallback','disp Hi!');
0048 %     javacomponent(b1);
0049 %
0050 %     f = figure;
0051 %     comp = awtcreate('javax.swing.JSpinner');
0052 %     [comp, container] = javacomponent(comp);
0053 %     set(container,'Position',[100, 100, 100, 40]);
0054 %     set(container,'Units','normalized');
0055 %
0056 %     f = figure;
0057 %     p = uipanel('Position',[0 0 .2 1]);
0058 %     ppos = getpixelposition(p);
0059 %     tr = awtcreate('javax.swing.JTree');
0060 %     [tree treecontainer] = javacomponent(tr, [0 0 ppos(3) ppos(4)], p);
0061 %
0062 %     f = figure('WindowState','docked');
0063 %     ta = awtcreate('javax.swing.JTable','II', 3, 10);
0064 %     table = javacomponent(ta, java.awt.BorderLayout.SOUTH, f);
0065 %
0066 %     f = figure;
0067 %     tb = uitoolbar(f);
0068 %     b2 = awtcreate('javax.swing.JButton','Ljava.lang.String;', 'Hi again!');
0069 %     b2 = javacomponent(b2, [], tb); % Note: Position is ignored.
0070 %     set(b2,'ActionPerformedCallback','disp('Hi again!'))');
0071 %
0072 % See also USEJAVACOMPONENT, AWTCREATE, AWTINVOKE
0073 %
0074 % Copyright 1984-2006 The MathWorks, Inc.
0075 % $Revision: 1.2 $ $Date: 2008/01/22 20:45:22 $
0076 %
0077 % Adapted for LTPDA
0078 %
0079 % $Id: ltpda_javacomponent.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0080 %
0081
0082 if (usejavacomponent == 0)
0083     err.message = 'JAVACOMPONENT is not supported on this platform';
0084     err.identifier = 'MATLAB:javacomponent:FeatureNotSupported';
0085     error(err);
0086 end
0087
0088 if ~isempty(nargchk(1,3,nargin))
0089     error('MATLAB:javacomponent',usage);
0090 end
0091
0092 if nargin < 3
0093     parent =(gcf);
0094 end
0095
0096 if nargin < 2
0097     position = [20 20 60 20];
0098 end
0099
0100 parentIsFigure = false;
0101 hParent = handle(parent);
0102 if ( isa(hParent, 'figure') || ...
0103     isa(hParent, 'uicontainer') || ...
0104     isa(hParent, 'uiflowcontainer') || ...
0105     isa(hParent, 'uigridcontainer'))
0106     parentIsFigure = true;
0107     peer = get(ancestor(parent,'figure'),'JavaFrame');
0108 elseif isa(hParent, 'uitoolbar')
0109     peer = get(parent,'JavaContainer');
0110     if isempty(peer)
0111         drawnow;
0112         peer = get(parent,'JavaContainer');
0113     end
0114 else
0115     error('MATLAB:javacomponent:InvalidParentHandle','Invalid parent handle\n%s',
usage)
0116 end
0117
0118 if isempty(peer)
0119     error('MATLAB:javacomponent:JavaFigsNotEnabled','Java figures are not enabled')
0120 end

```

```

0121
0122 hUicontainer = [];
0123 hgp = [];
0124 returnContainer = 1;
0125
0126 if ischar(component)
0127     component = awtcreate(component);
0128 end
0129
0130 % promote the component to a handle object first.
0131 % It seems once a java object is cast to a handle, you cannot get another
0132 % handle with 'callbackproperties'. So, do want you need outright and use
0133 % the same handle in the subsequent code.
0134 if ~isjava(component)
0135     component = java(component);
0136 end
0137 hcomponent = handle(component, 'callbackProperties');
0138
0139 if nargin == 1
0140     hgp = handle(peer.addchild(component));
0141     % parent must be a figure, we default to(gcf upstairs
0142     createPanel;
0143     hgp.setUIContainer(hUicontainer);
0144 else
0145     if parentIsFigure
0146         if isnumeric(position)
0147             if isempty(position)
0148                 position = [20 20 60 20];
0149             end
0150             % numeric position is not set here, rely on the uicontainer
0151             % listeners below.
0152             hgp = handle(peer.addchild(component));
0153             createPanel;
0154             hgp.setUIContainer(hUicontainer);
0155         elseif ...
0156             isequal(char(position),char(java.awt.BorderLayout.NORTH)) || ...
0157             isequal(char(position),char(java.awt.BorderLayout.SOUTH)) || ...
0158             isequal(char(position),char(java.awt.BorderLayout.EAST)) || ...
0159             isequal(char(position),char(java.awt.BorderLayout.WEST)) || ...
0160             hgp = handle(peer.addchild(component, position));
0161             returnContainer = 0;
0162             createPanel;
0163         else
0164             error('MATLAB:javacomponent:InvalidPosition', 'Invalid component
0165 position\n%s', usage)
0166         end
0167     else
0168         % Adding component to the toolbar.
0169         % component position is ignored for now
0170         peer.add(component);
0171         hUicontainer = parent; % toolbar.
0172         handles = getappdata(hUicontainer, 'childhandles');
0173         handles = [handles, hcomponent];
0174         setappdata(hUicontainer, 'childhandles', handles);
0175     end
0176     % make sure the component is on the screen so the
0177     % caller can interact with it right now.
0178     % drawnow;
0179 end
0180 configureComponent;
0181
0182 if (returnContainer == 1)
0183     hcontainer = hUicontainer;
0184 else
0185     hcontainer = [];
0186 end
0187
0188 function createPanel
0189     % add delete listener
0190     hUicontainer = hgjavacomponent('Parent', parent, 'Units', 'Pixels');
0191     set(hUicontainer, 'UserData', char(component.getClass.getName)); % For findobj
0192 queries.
0193     if isa(java(hgp), 'com.mathworks.hg.peer.FigureChild')
0194         set(hUicontainer, 'FigureChild', hgp);
0195     end
0196     if isa(java(hcomponent), 'javax.swing.JComponent')
0197         % force component to be opaque if it's a JComponent. This prevents
0198         % lightweight component from showing the figure background (which
0199         % may never get a paint event)
0200         hcomponent.setOpaque(true);
0201     end
0202     set(hUicontainer, 'JavaPeer', hcomponent);
0203
0204

```



```

0205         if (returnContainer == 1)
0206             % add resize listener to parent (parent must be a figure or this dies
quietly)
0207             % this is for normalized units
0208             addlistener(hUicontainer, 'PixelBounds', 'PropertyPostSet', @handleResize);
0209
0210             % add visible listener
0211             addlistener(hUicontainer, 'Visible', 'PropertyPostSet', @handleVisible);
0212
0213             % add parent listener
0214             addlistener(hUicontainer, 'Parent', 'PropertyPreSet', @handlePreParent);
0215
0216             % add parent listener
0217             addlistener(hUicontainer, 'Parent', 'PropertyPostSet', @handlePostParent);
0218
0219             % force though 1st resize event
0220             set(hUicontainer, 'Position', position);
0221         else
0222             % For the BorderLayout components, we dont really want the
0223             % hUicontainer to show. But, it provides a nice place for cleanup.
0224             set(hUicontainer, 'Visible', 'off', 'Handlevisibility', 'off');
0225             % Set position out of the figure to work around a current bug
0226             % due to which invisible uicontainers show up when renderer is
0227             % OpenGL (G.
0228             set(hUicontainer, 'Position', [1 1 0.01 0.01]);
0229         end
0230
0231         if isa(component, 'com.mathworks.hg.peer.FigureChild')
0232             component.setUIContainer(hUicontainer);
0233         end
0234
0235         function handleResize(obj, evd) %#ok - mlint
0236             hgp.setPixelPosition(getpixelposition(hUicontainer, true));
0237         end
0238
0239         function handleVisible(obj, evd) %#ok - mlint
0240             hgp.setVisible(strcmp(get(hUicontainer, 'Visible'), 'on'))
0241         end
0242
0243         function handlePreParent(obj, evd) %#ok - mlint
0244             oldfig = ancestor(parent, 'figure');
0245             newfig = ancestor(evd.NewValue, 'figure');
0246             if ~isempty(newfig) && ~isequal(oldfig, newfig)
0247                 peer = get(oldfig, 'JavaFrame');
0248                 peer.remove(component);
0249             end
0250         end
0251
0252         function handlePostParent(obj, evd) %#ok - mlint
0253             oldfig = ancestor(parent, 'figure');
0254             newfig = ancestor(evd.NewValue, 'figure');
0255             if ~isempty(newfig) && ~isequal(oldfig, newfig)
0256                 peer = get(newfig, 'JavaFrame');
0257                 hgp = handle(peer.addchild(component));
0258                 if isa(java(hgp), 'com.mathworks.hg.peer.FigureChild')
0259                     % used by the uicontainer C-code
0260                     setappdata(hUicontainer, 'FigureChild', java(hgp));
0261                 end
0262                 parent = newfig;
0263             end
0264             hgp.setPixelPosition(getpixelposition(hUicontainer, true));
0265         end
0266     end
0267
0268     function configureComponent
0269         set(hUicontainer, 'DeleteFcn', {@containerDelete, hcomponent});
0270         addlistener(hcomponent, 'ObjectBeingDestroyed', {@componentDelete, hUicontainer,
parentIsFigure});
0271     end
0272
0273 end
0274
0275 function containerDelete(obj, evd, hc) %#ok - mlint
0276     if isa(handle(obj), 'uitoolbar')
0277         childHandles = getappdata(obj, 'childhandles');
0278         delete(childHandles);
0279     else
0280         if ishandle(hc)
0281             delete(hc);
0282         end
0283     end
0284 end
0285
0286 function componentDelete(obj, evd, hUicontainer, parentIsFigure) %#ok - mlint
0287 if (parentIsFigure)
0288     % This java component is always deleted before hUicontainer. It is

```

```

0289 % ensured by calling component deletion in function containerDelete.
0290 % hUicontainer becomes invalid when delete(hUicontainer) below is run.
0291 parent = ancestor(hUicontainer, 'figure');
0292 peer = get(parent, 'JavaFrame');
0293
0294 if any(ishandle(obj))
0295     removeobj = java(obj);
0296     if ~isempty(get(hUicontainer, 'FigureChild'))
0297         removeobj = get(hUicontainer, 'FigureChild');
0298     end
0299     peer.remove(removeobj);
0300 end
0301
0302 % delete container if it exists
0303 if any(ishandle(hUicontainer))
0304     delete(hUicontainer);
0305 end
0306 else
0307     parent = hUicontainer; % toolbar
0308     if ~ishandle(parent) || ~ishandle(obj)
0309         % The toolbar parent or the component has been deleted. Bail out.
0310         % Toolbar clears all javacomponents after itself.
0311         return;
0312     end
0313
0314     peer = get(parent, 'JavaContainer');
0315     if ~isempty(peer)
0316         peer.remove(java(obj));
0317     end
0318 end
0319 end
0320
0321
0322 function str=usage
0323 str = [sprintf('\n') ...
0324     'WARNING: This feature is not supported in MATLAB and the API and ' sprintf('\n')
0325     'functionality may change in a future release. ' sprintf('\n') ...
0326 ];
0327 %{
0328 'usage: javacomponent(hJavaComponent, position, parent) ' sprintf('\n') ...
0329 '- position can be [left bottom width height] in pixels ' sprintf('\n') ...
0330 '- or the string North, South, East, or West' sprintf('\n') ...
0331 '- parent can be a figure or a uitoolbar handle.'];
0332 %}
0333 end

```

[Home](#) > [m](#) > [gui](#) > [ao_browser](#) > ltpda_uitree.m

ltpda_uitree

PURPOSE ^

WARNING: This feature is not supported in MATLAB

SYNOPSIS ^

function [tree, container] = ltpda_uitree(varargin)

DESCRIPTION ^

WARNING: This feature is not supported in MATLAB and the API and functionality may change in a future release.

UITREE creates a uitree component with hierarchical data in a figure window. UITREE creates an empty uitree object with default property values in a figure window.

UITREE('PropertyName1', 'Value1', 'PropertyName2', 'Value2', ...) creates a uitree object with the specified properties. The properties that can be set are: Root, ExpandFcn, SelectionChangeFcn, Parent and Position. The 'Root' property must be specified to successfully to create a uitree. The other properties are optional.

UITREE(figurehandle, ...) creates a uitree object in the figure window specified by the figurehandle.

HANDLE = UITREE(...) creates a uitree object and returns its handle.

Properties:

Root - Root node for the uitree object. Could be handle to a HG object, a string, an open block diagram name, or handle to a UITREENODE object.

ExpandFcn - Node expansion function. String or function handle.

SelectionChangeFcn - Selection callback function. String or function handle.

Parent - Parent figure handle. If not specified, it is the gcf.

Position: 4 element vector specifying the position.

DndEnabled: Boolean specifying if drag and drop is enabled (false).

MultipleSelectionEnabled: Boolean specifying if multiple selection is allowed (false).

SelectedNodes: vector of uitreenodes to be selected.

Units: String - pixels/normalized/inches/points/centimeters.

Visible: Boolean specifying if table is visible.

NodeDroppedCallback: Callback for a drag and drop action.

NodeExpandedCallback: Callback for a node expand action.

NodeCollapsedCallback: Callback function for a node collapse action.

NodeSelectedCallback: Callback for a node selection action.

Examples:

```
t = uitree('Root', 'D:\')
```

```
%Creates a uitree widget in a figure window with which acts as a
%directory browser with the D: drive as the root node.
```

```
surf(peaks)
f = figure
t = uitree(f, 'Root', 0)
```

```
%Creates a uitree object in the specified figure window which acts as
%a MATLAB hierarchy browser with the MATLAB root (0) as the root node.
```

```
root = uitreenode('S:\', 'S', [], false);
t = uitree('Root', root, 'ExpandFcn', @myExpFcn, ...
    'SelectionChangeFcn', 'disp(''Selection Changed'')');
```

```
%Creates a uitree object with the specified root node and a custom
%function to return child nodes for any given node. The function
%myExpFcn is an m-file in the MATLAB path with the following code:
```

```
%This function should be added to your path
% -----
function nodes = myExpFcn(tree, value)

try
    count = 0;
    ch = dir(value);
    for i=1:length(ch)
        if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)
            count = count + 1;
            if ch(i).isdir
                iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
            else
                iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.gif'];
            end
            nodes(count) = uitreenode([value, ch(i).name, filesep], ...
                ch(i).name, iconpath, ~ch(i).isdir);
        end
    end
catch
    error(['The uitree node type is not recognized. You may need to ', ...
        'define an ExpandFcn for the nodes.']);
end

if (count == 0)
    nodes = [];
end
% -----
```

See also UITREENODE, UITABLE, PATH

Copyright 2003-2006 The MathWorks, Inc.
 \$Revision: 1.2 \$ \$Date: 2008/01/22 20:45:22 \$

Adapted for LTPDA

\$Id: ltpda_uitree.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [ltpda_explorer](#) LTPDA_EXPLORER for exploring analysis objects and plotting/displaying their fields

SUBFUNCTIONS [^]

-  [function nodeExpanded\(src, evd, tree, expfcn\)](#)
-  [function nodeSelected\(src, evd, tree, selfcn\)](#)
-  [function \[node, expfcn\] = processNode\(root\)](#)
-  [function nodes = hgBrowser\(tree, value\)](#)
-  [function nodes = mdlBrowser\(tree, value\)](#)
-  [function nodes = dirBrowser\(tree, value\)](#)
-  [function yesno = ismodel\(input\)](#)

SOURCE CODE [^]

```
0001 function [tree, container] = ltpda_uitree(varargin)
0002 % WARNING: This feature is not supported in MATLAB
0003 % and the API and functionality may change in a future release.
0004 %
0005 % UITREE creates a uitree component with hierarchical data in a figure window.
0006 % UITREE creates an empty uitree object with default property values in
0007 % a figure window.
0008 %
0009 % UITREE('PropertyName1', 'Value1', 'PropertyName2', 'Value2', ...)
0010 % creates a uitree object with the specified properties. The properties
0011 % that can be set are: Root, ExpandFcn, SelectionChangeFcn, Parent and
0012 % Position. The 'Root' property must be specified to successfully to
0013 % create a uitree. The other properties are optional.
```

```

0014 %
0015 % UITREE(handle, ...) creates a uitree object in the figure
0016 % window specified by the handle.
0017 %
0018 % HANDLE = UITREE(...) creates a uitree object and returns its handle.
0019 %
0020 % Properties:
0021 %
0022 % Root - Root node for the uitree object. Could be handle to a HG
0023 % object, a string, an open block diagram name, or handle to a
0024 % UITREENODE object.
0025 % ExpandFcn - Node expansion function. String or function handle.
0026 % SelectionChangeFcn - Selection callback function. String or function
0027 % handle.
0028 % Parent - Parent figure handle. If not specified, it is the gcf.
0029 % Position: 4 element vector specifying the position.
0030 %
0031 % DndEnabled: Boolean specifying if drag and drop is enabled (false).
0032 % MultipleSelectionEnabled: Boolean specifying if multiple selection is
0033 % allowed (false).
0034 % SelectedNodes: vector of uitreenodes to be selected.
0035 % Units: String - pixels/normalized/inches/points/centimeters.
0036 % Visible: Boolean specifying if table is visible.
0037 % NodeDroppedCallback: Callback for a drag and drop action.
0038 % NodeExpandedCallback: Callback for a node expand action.
0039 % NodeCollapsedCallback: Callback function for a node collapse action.
0040 % NodeSelectedCallback: Callback for a node selection action.
0041 %
0042 %
0043 % Examples:
0044 %     t = uitree('Root', 'D:\')
0045 %
0046 %     %Creates a uitree widget in a figure window with which acts as a
0047 %     %directory browser with the D: drive as the root node.
0048 %
0049 %     surf(peaks)
0050 %     f = figure
0051 %     t = uitree(f, 'Root', 0)
0052 %
0053 %     %Creates a uitree object in the specified figure window which acts as
0054 %     %a MATLAB hierarchy browser with the MATLAB root (0) as the root node.
0055 %
0056 %     root = uitreenode('S:\', 'S', [], false);
0057 %     t = uitree('Root', root, 'ExpandFcn', @myExpFcn, ...
0058 %         'SelectionChangeFcn', 'disp(''Selection Changed'')');
0059 %
0060 %     %Creates a uitree object with the specified root node and a custom
0061 %     %function to return child nodes for any given node. The function
0062 %     %myExpFcn is an m-file in the MATLAB path with the following code:
0063 %
0064 %     %This function should be added to your path
0065 %     % -----
0066 %     function nodes = myExpFcn(tree, value)
0067 %
0068 %     try
0069 %         count = 0;
0070 %         ch = dir(value);
0071 %
0072 %         for i=1:length(ch)
0073 %             if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)
0074 %                 count = count + 1;
0075 %                 if ch(i).isdir
0076 %                     iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
0077 %                 else
0078 %                     iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.gif'];
0079 %                 end
0080 %                 nodes(count) = uitreenode([value, ch(i).name, filesep], ...
0081 %                     ch(i).name, iconpath, ~ch(i).isdir);
0082 %             end
0083 %         end
0084 %     catch
0085 %         error(['The uitree node type is not recognized. You may need to ', ...
0086 %             'define an ExpandFcn for the nodes.']);
0087 %     end
0088 %
0089 %     if (count == 0)
0090 %         nodes = [];
0091 %     end
0092 %     % -----
0093 %
0094 % See also UITREENODE, UITABLE, PATH
0095 %
0096 % Copyright 2003-2006 The MathWorks, Inc.
0097 % $Revision: 1.2 $ $Date: 2008/01/22 20:45:22 $
0098 %
0099 % Adapted for LTPDA

```

```

0100 %
0101 % $Id: ltpda UITree.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0102 %
0103
0104 %   Release: R14. This feature will not work in previous versions of MATLAB.
0105
0106 %% Setup and P-V parsing.
0107
0108 error(javachk('awt'));
0109 error(nargoutchk(0, 2, nargout));
0110
0111 fig = [];
0112 numargs = nargin;
0113
0114 if (nargin > 0 && isscalar(varargin{1}) && ishandle(varargin{1}))
0115     if ~isa(handle(varargin{1}), 'figure')
0116         error('MATLAB:UITree:InvalidFigureHandle', 'Unrecognized parameter.');
```

```

0117     end
0118     fig = varargin{1};
0119     varargin = varargin(2:end);
0120     numargs = numargs - 1;
0121 end
0122
0123 % RootFound = false;
0124 root = [];
0125 expfcn = [];
0126 selfcn = [];
0127 pos = [];
0128 % parent = [];
0129
0130 if (numargs == 1)
0131     error('MATLAB:UITree:InvalidNumInputs', 'Unrecognized parameter.');
```

```

0132 end
0133
0134 for i = 1:2:numargs-1
0135     if ~ischar(varargin{i})
0136         error('MATLAB:UITree:UnrecognizedParameter', 'Unrecognized parameter.');
```

```

0137     end
0138     switch lower(varargin{i})
0139     case 'root'
0140         root = varargin{i+1};
0141     case 'expandfcn'
0142         expfcn = varargin{i+1};
0143     case 'selectionchangeFcn'
0144         selfcn = varargin{i+1};
0145     case 'parent'
0146         if ishandle(varargin{i+1})
0147             f = varargin{i+1};
0148             if isa(handle(f), 'figure')
0149                 fig = f;
0150             end
0151         end
0152     case 'position'
0153         p = varargin{i+1};
0154         if isnumeric(p) && (length(p) == 4)
0155             pos = p;
0156         end
0157     otherwise
0158         error('MATLAB:UITree:UnknownParameter', ['Unrecognized parameter: ',
0159 varargin{i}]);
0160     end
0161 end
0162
0163 if isempty(expfcn)
0164     [root, expfcn] = processNode(root);
0165 else
0166     root = processNode(root);
0167 end
0168
0169 tree_h = com.mathworks.hg.peer.UITreePeer;
0170 tree_h.setRoot(root);
0171
0172 if isempty(fig)
0173     fig = gcf;
0174 end
0175
0176 if isempty(pos)
0177     figpos = get(fig, 'Position');
0178     pos = [0 0 min(figpos(3), 200) figpos(4)];
0179 end
0180
0181 % pass the figure child in, let javacomponent introspect
0182 [obj, container] = javacomponent(tree_h, pos, fig);
0183 % javacomponent returns a UDD handle for the java component passed in.
0184 tree = obj;

```



```

0185
0186 if ~isempty(expfcn)
0187     set(tree, 'NodeExpandedCallback', {@nodeExpanded, tree, expfcn});
0188 end
0189
0190 if ~isempty(selfcn)
0191     set(tree, 'NodeSelectedCallback', {@nodeSelected, tree, selfcn});
0192 end
0193
0194 end
0195
0196 %% -----
0197 function nodeExpanded(src, evd, tree, expfcn)                                %#ok
0198
0199 % tree = handle(src);
0200 % evdsrc = evd.getSource;
0201
0202 evdnode = evd.getCurrentNode;
0203 % indices = [];
0204
0205 if ~tree.isLoaded(evdnode)
0206     value = evdnode.getValue;
0207
0208     % <call a user function(value) which returns uitreenodes>;
0209     cbk = expfcn;
0210     if iscell(cbk)
0211         childnodes = feval(cbk{1}, tree, value, cbk{2:end});
0212     else
0213         childnodes = feval(cbk, tree, value);
0214     end
0215
0216     if (length(childnodes) == 1)
0217         % Then we dont have an array of nodes. Create an array.
0218         chnodes = childnodes;
0219         childnodes = javaArray('com.mathworks.hg.peer.UITreeNode', 1);
0220         childnodes(1) = java(chnodes);
0221     end
0222
0223     tree.add(evdnode, childnodes);
0224     tree.setLoaded(evdnode, true);
0225 end
0226
0227 end
0228
0229 %% -----
0230 function nodeSelected(src, evd, tree, selfcn)                                %#ok
0231 cbk = selfcn;
0232 hgfeval(cbk, tree, evd);
0233
0234 end
0235
0236 %% -----
0237 function [node, expfcn] = processNode(root)
0238 expfcn = [];
0239
0240 if isempty(root) || isa(root, 'com.mathworks.hg.peer.UITreeNode') || ...
0241     isa(root, 'javahandle.com.mathworks.hg.peer.UITreeNode')
0242     node = root;
0243 elseif ishghandle(root)
0244     % Try to process as an HG object.
0245     try
0246         node = uitreenode(handle(root), get(root, 'Type'), ...
0247             [], isempty(get(0, 'children')));
0248     catch
0249         node = [];
0250     end
0251     expfcn = @hgBrowser;
0252 elseif ismodel(root)
0253     % Try to process as an open Simulink system
0254
0255     % TODO if there is an open simulink system and a directory on the path with
0256     % the same name, the system will hide the directory. Perhaps we should
0257     % warn about this.
0258     try
0259         h = handle(get_param(root, 'Handle'));
0260         % TODO we pass root to the tree as a string,
0261         % it would be better if we could just pass the
0262         % handle up
0263         node = uitreenode(root, get(h, 'Name'), ...
0264             [], isempty(h.getHierarchicalChildren));
0265     catch
0266         node = [];
0267     end
0268     expfcn = @mdlBrowser;
0269 elseif ischar(root)
0270     % Try to process this as a directory structure.

```

```

0271     try
0272         iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
0273         node = uitreenode(root, root, iconpath, ~isdir(root));
0274     catch
0275         node = [];
0276     end
0277     expfcn = @dirBrowser;
0278 else
0279     node = [];
0280 end
0281 end
0282 end
0283
0284 %% -----
0285 function nodes = hgBrowser(tree, value)                                %#ok
0286
0287 try
0288     count = 0;
0289     parent = handle(value);
0290     ch = parent.children;
0291     for i=1:length(ch)
0292         count = count+1;
0293         nodes(count) = uitreenode(handle(ch(i)), get(ch(i), 'Type'), [], ...
0294             isempty(get(ch(i), 'children')));
0295     end
0296 catch
0297     error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not recognized. You
0298 may need to ' '...',
0299         'define an ExpandFcn for the nodes.']);
0300 end
0301
0302 if (count == 0)
0303     nodes = [];
0304 end
0305
0306 end
0307
0308 %% -----
0309 function nodes = mdlBrowser(tree, value)                                %#ok
0310
0311 try
0312     count = 0;
0313     parent = handle(get_param(value, 'Handle'));
0314     ch = parent.getHierarchicalChildren;
0315     for i=1:length(ch)
0316         if isempty(findstr(class(ch(i)), 'SubSystem'))
0317             % not a subsystem
0318         else
0319             % is a subsystem
0320             count = count+1;
0321             descr = get(ch(i), 'Name');
0322             isleaf = true;
0323             cch = ch(i).getHierarchicalChildren;
0324             if ~isempty(cch)
0325                 for j = 1:length(cch)
0326                     if ~isempty(findstr(class(cch(j)), 'SubSystem'))
0327                         isleaf = false;
0328                         break;
0329                     end
0330                 end
0331             end
0332             nodes(count) = uitreenode([value '/' descr], descr, [], ...
0333                 isleaf);
0334         end
0335     end
0336 catch
0337     error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not recognized. You
0338 may need to ' '...',
0339         'define an ExpandFcn for the nodes.']);
0340 end
0341
0342 if (count == 0)
0343     nodes = [];
0344 end
0345
0346 end
0347
0348 %% -----
0349 function nodes = dirBrowser(tree, value)                                %#ok
0350
0351 try
0352     count = 0;
0353     ch = dir(value);

```



```

0355
0356     for i=1:length(ch)
0357         if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)
0358             count = count + 1;
0359             if ch(i).isdir
0360                 iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
0361             else
0362                 iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.gif'];
0363             end
0364             nodes(count) = uitreenode([value, ch(i).name, filesep], ...
0365                                     ch(i).name, iconpath, ~ch(i).isdir);
0366         end
0367     end
0368 catch
0369     error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not recognized. You
may need to '
0370         'define an ExpandFcn for the nodes.']);
0371 end
0372
0373 if (count == 0)
0374     nodes = [];
0375 end
0376
0377 end
0378
0379 %% -----
0380 function yesno = ismodel(input)
0381 yesno = false;
0382
0383 try
0384     get_param(input, 'handle');
0385     yesno = true;
0386 catch
0387 end
0388
0389 end

```

[Home](#) > [m](#) > [gui](#) > [ao_browser](#) > [Itpda_uitreenode.m](#)

Itpda_uitreenode

PURPOSE ^

WARNING: This feature is not supported in MATLAB

SYNOPSIS ^

function node = Itpda_uitreenode(value, string, icon, isLeaf)

DESCRIPTION ^

WARNING: This feature is not supported in MATLAB and the API and functionality may change in a future release.

UITREENODE(Value, Description, Icon, Leaf)
creates a tree node object for the uitree with the specified properties. All properties must be specified for the successful creation of a node object.

Value can be a string or handle represented by this node.
Description is a string which is used to identify the node.
Icon can be a qualified pathname to an image to be used as an icon for this node. It may be set to [] to use default icons.
Leaf can be true or false to denote whether this node has children.

Example:

```
t = uitree('Root', 'D:\')
```

```
%Creates a uitree widget in a figure window with which acts as a  
%directory browser with the D: drive as the root node.
```

```
surf(peaks)  
f = figure  
t = uitree(f, 'Root', 0)
```

```
%Creates a uitree object in the specified figure window which acts as  
%a MATLAB hierarchy browser with the MATLAB root (0) as the root node.
```

```
root = uitreenode('S:\', 'S', [], false);  
t = uitree('Root', root, 'ExpandFcn', @myExpFcn, ...  
    'SelectionChangeFcn', 'disp(''Selection Changed'')');
```

```
%Creates a uitree object with the specified root node and a custom  
%function to return child nodes for any given node. The function  
%myExpFcn is a user defined m-file in the MATLAB path.
```

```
% This function should be added to the path  
% -----  
function nodes = myExpFcn(tree, value)
```

```
try  
    count = 0;  
    ch = dir(value);  
    for i=1:length(ch)  
        if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)  
            count = count + 1;  
            if ch(i).isdir  
                iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];  
            else  
                iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.gif'];  
            end  
            nodes(count) = uitreenode([value, ch(i).name, filesep], ...  
                ch(i).name, iconpath, ~ch(i).isdir);  
        end  
    end  
catch  
    error(['The uitree node type is not recognized. You may need to ', ...  
        'define an ExpandFcn for the nodes.']);  
end  
  
if (count == 0)  
    nodes = [];
```

```
end
% -----
```

See also UITREE, UITABLE, JAVACOMPONENT

Copyright 2003-2006 The MathWorks, Inc.

Adapted for LTPDA

\$Id: ltpda_uitreenode.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_explorer](#) LTPDA_EXPLORER for exploring analysis objects and plotting/displaying their fields

SOURCE CODE ^

```
0001 function node = ltpda_uitreenode(value, string, icon, isLeaf)
0002 % WARNING: This feature is not supported in MATLAB
0003 % and the API and functionality may change in a future release.
0004 %
0005 %     UITREENODE(Value, Description, Icon, Leaf)
0006 %     creates a tree node object for the uitree with the specified
0007 %     properties. All properties must be specified for the successful
0008 %     creation of a node object.
0009 %
0010 %     Value can be a string or handle represented by this node.
0011 %     Description is a string which is used to identify the node.
0012 %     Icon can be a qualified pathname to an image to be used as an icon
0013 %     for this node. It may be set to [] to use default icons.
0014 %     Leaf can be true or false to denote whether this node has children.
0015 %
0016 %     Example:
0017 %         t = uitree('Root', 'D:\')
0018 %
0019 %         %Creates a uitree widget in a figure window with which acts as a
0020 %         %directory browser with the D: drive as the root node.
0021 %
0022 %         surf(peaks)
0023 %         f = figure
0024 %         t = uitree(f, 'Root', 0)
0025 %
0026 %         %Creates a uitree object in the specified figure window which acts as
0027 %         %a MATLAB hierarchy browser with the MATLAB root (0) as the root node.
0028 %
0029 %         root = uitreenode('S:\', 'S', [], false);
0030 %         t = uitree('Root', root, 'ExpandFcn', @myExpFcn, ...
0031 %             'SelectionChangeFcn', 'disp(''Selection Changed'')');
0032 %
0033 %         %Creates a uitree object with the specified root node and a custom
0034 %         %function to return child nodes for any given node. The function
0035 %         %myExpFcn is a user defined m-file in the MATLAB path.
0036 %
0037 %         % This function should be added to the path
0038 %         % -----
0039 %         function nodes = myExpFcn(tree, value)
0040 %
0041 %         try
0042 %             count = 0;
0043 %             ch = dir(value);
0044 %
0045 %             for i=1:length(ch)
0046 %                 if (any(strcmp(ch(i).name, {'.', '..', ''})) == 0)
0047 %                     count = count + 1;
0048 %                     if ch(i).isdir
0049 %                         iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
0050 %                     else
0051 %                         iconpath = [matlabroot, '/toolbox/matlab/icons/pageicon.gif'];
0052 %                     end
0053 %                     nodes(count) = uitreenode([value, ch(i).name, filesep], ...
0054 %                         ch(i).name, iconpath, ~ch(i).isdir);
0055 %                 end
0056 %             end
0057 %         catch
0058 %             error(['The uitree node type is not recognized. You may need to ', ...
```

```

0059 %           'define an ExpandFcn for the nodes.']);
0060 %       end
0061 %
0062 %       if (count == 0)
0063 %           nodes = [];
0064 %       end
0065 %       % -----
0066 %
0067 %       See also UITREE, UITABLE, JAVACOMPONENT
0068 %
0069 % Copyright 2003-2006 The MathWorks, Inc.
0070 %
0071 % Adapted for LTPDA
0072 %
0073 % $Id: ltpda_uitreenode.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0074 %
0075
0076 import com.mathworks.hg.peer.UITreeNode;
0077 node = handle(UITreeNode(value, string, icon, isLeaf));
0078 schema.prop(node, 'UserData', 'MATLAB array');

```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

Contents

PURPOSE ^

%% path: m/gui/constructor_helper
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/constructor_helper  %%%%%%%%%%%%%%  
    <a href="matlab:help  
m/gui/constructor_helper/ltpda_constructor_helper">m/gui/constructor_helper/ltpda_constructor_he  
- allows the user to explore object constructors.  
    <a href="matlab:help  
m/gui/constructor_helper/ltpda_constructor_helper_build">m/gui/constructor_helper/ltpda_construc  
- Callback for the build constructor gui  
    <a href="matlab:help  
m/gui/constructor_helper/ltpda_constructor_helper_buildParams">m/gui/constructor_helper/ltpda_co  
- parameter panel on the main figure  
    <a href="matlab:help  
m/gui/constructor_helper/ltpda_constructor_helper_class">m/gui/constructor_helper/ltpda_construc  
- Callback executed when user selects a class  
    <a href="matlab:help  
m/gui/constructor_helper/ltpda_constructor_helper_sets">m/gui/constructor_helper/ltpda_construc  
- Callback executed when user selects a parameter set
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpda_constructor_helper

PURPOSE ^

LTPDA_CONSTRUCTOR_HELPER allows the user to explore object constructors.

SYNOPSIS ^

function varargout = ltpda_constructor_helper(varargin)

DESCRIPTION ^

```
LTPDA_CONSTRUCTOR_HELPER allows the user to explore object constructors.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_CONSTRUCTOR_HELPER allows the user to explore object constructors.
CALL:       ltpda_constructor_helper

VERSION:     $Id: ltpda_constructor_helper.m,v 1.4 2008/06/20 10:51:55 hewitson Exp $
HISTORY:     07-03-08 M Hewitson
              Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

 [function ltpda_constructor_helper_close\(varargin\)](#)

SOURCE CODE ^

```
0001 function varargout = ltpda_constructor_helper(varargin)
0002
0003 % LTPDA_CONSTRUCTOR_HELPER allows the user to explore object constructors.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: LTPDA_CONSTRUCTOR_HELPER allows the user to explore object constructors.
0008 %
0009 % CALL:       ltpda_constructor_helper
0010 %
0011 %
0012 % VERSION:     $Id: ltpda_constructor_helper.m,v 1.4 2008/06/20 10:51:55 hewitson Exp $
0013 %
0014 % HISTORY:     07-03-08 M Hewitson
0015 %              Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019
0020 %% Check if I exist already
0021 id = findobj('Tag', 'LTPDAconstructor_helper');
0022 if ~isempty(id)
0023     figure(id)
0024     return
0025 end
```

```

0026
0027 %% Some initial setup
0028
0029 Gproperties.Gcol = [240 240 240]/255;
0030 Gproperties.Gwidth = 500;
0031 Gproperties.Gheight = 300;
0032 Gproperties.Gborder = 10;
0033 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0034
0035 Gproperties.Screen = get(0, 'screensize');
0036 Gproperties.Gposition = [150 ...
0037     150 ...
0038     Gproperties.Gwidth...
0039     Gproperties.Gheight];
0040
0041 % Initialize and hide the GUI as it is being constructed.
0042 mainfig = figure('Name', 'LTPDA Constructor Helper',...
0043     'NumberTitle', 'off',...
0044     'Visible', 'off',...
0045     'Position', Gproperties.Gposition,...
0046     'Color', Gproperties.Gcol,...
0047     'Resize', 'off',...
0048     'ToolBar', 'none',...
0049     'NextPlot', 'new',...
0050     'MenuBar', 'none',...
0051     'Tag', 'LTPDAconstructor_helper');
0052
0053 % Set mainfig callbacks
0054 set(mainfig, 'CloseRequestFcn', {@ltpda_constructor_helper_close, mainfig});
0055
0056 % Set Application data
0057 setappdata(mainfig, 'Gproperties', Gproperties);
0058
0059
0060 %% GUI parts
0061
0062 %--- Class selection
0063 % text field
0064 sth = uicontrol(mainfig, 'Style', 'text',...
0065     'String', 'Class',...
0066     'HorizontalAlignment', 'left',...
0067     'BackgroundColor', Gproperties.Gcol,...
0068     'Position', [10 Gproperties.Gheight-40 50 25]);
0069
0070 % pop-up dialog
0071 pmh = uicontrol(mainfig, 'Style', 'popupmenu',...
0072     'String', utils.helper.ltpda_userclasses,...
0073     'Value', 1,...
0074     'BackgroundColor', Gproperties.Gcol,...
0075     'Tag', 'LTPDA_constructor_helper_class',...
0076     'Callback', {'ltpda_constructor_helper_class', mainfig},...
0077     'Position', [90 Gproperties.Gheight-40 120 25]);
0078
0079 %--- Constructor set list
0080
0081
0082 % text field
0083 sth = uicontrol(mainfig, 'Style', 'text',...
0084     'String', 'Sets',...
0085     'HorizontalAlignment', 'left',...
0086     'BackgroundColor', Gproperties.Gcol,...
0087     'Position', [10 Gproperties.Gheight-80 50 25]);
0088
0089 % Set list
0090 lbh = uicontrol(mainfig, 'Style', 'listbox',...
0091     'String', '',...
0092     'Value', 1,...
0093     'BackgroundColor', 'w',...
0094     'Tag', 'LTPDA_constructor_helper_sets',...
0095     'Callback', {'ltpda_constructor_helper_sets', mainfig},...
0096     'Position', [10 Gproperties.Gheight-270 200 200]);
0097
0098 % Constructor view
0099 eth = uicontrol(mainfig, 'Style', 'edit',...
0100     'String', '',...
0101     'BackgroundColor', 'w',...
0102     'Max', 100,...
0103     'HorizontalAlignment', 'left',...
0104     'FontSize', 14,...
0105     'Tag', 'LTPDA_constructor_helper_constructor',...
0106     'Position', [240 Gproperties.Gheight-210 250 200]);
0107
0108 % Variable name
0109 sth = uicontrol(mainfig, 'Style', 'text',...
0110     'String', 'Variable',...
0111     'HorizontalAlignment', 'left',...

```



```

0112     'BackgroundColor', Gproperties.Gcol, ...
0113     'Position',[240 Gproperties.Gheight-260 50 25]);
0114
0115 eth = uicontrol(mainfig,'Style','edit',...
0116     'String','obj',...
0117     'BackgroundColor','w',...
0118     'HorizontalAlignment','center',...
0119     'Tag','LTPDA_constructor_helper_var',...
0120     'Position',[300 Gproperties.Gheight-260 50 25]);
0121
0122
0123 % Build btn
0124 pbh = uicontrol(mainfig,'Style','pushbutton','String','Build',...
0125     'Position',[360 Gproperties.Gheight-260 60 25],...
0126     'Tag','LTPDA_constructor_helper_buildBtn');
0127 set(pbh, 'Callback', {'ltpda_constructor_helper_build'});
0128
0129 %% Start the GUI
0130
0131 % Make the GUI visible.
0132 set(mainfig,'Visible','on')
0133
0134
0135 %% Callbacks
0136
0137 %--- Close
0138 function ltpda_constructor_helper_close(varargin)
0139     disp('* Goodbye from the LTPDA Constructor Helper *')
0140     delete(varargin{end})
0141 end
0142 end
0143
0144
0145

```

[Home](#) > [m](#) > [gui](#) > [constructor_helper](#) > ltpda_constructor_helper_build.m

ltpda_constructor_helper_build

PURPOSE ^

Callback for the build constructor gui

SYNOPSIS ^

function ltpda_constructor_helper_build(varargin)

DESCRIPTION ^

Callback for the build constructor gui

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```

0001 function ltpda_constructor_helper_build(varargin)
0002 % Callback for the build constructor gui
0003
0004 mainfig = findobj('Tag', 'LTPDAconstructor_helper');
0005
0006 % get object name
0007 varh = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_var');
0008 varname = get(varh, 'String');
0009
0010 % get selected class
0011 clh = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
0012 cltxt = get(clh, 'String');
0013 cl = cltxt{get(clh, 'Value')};
0014
0015 % get base workspace variables
0016 ws_vars = evalin('base', 'who');
0017
0018 % check if this one is there or not
0019 if ismember(varname, ws_vars)
0020     j = 1;
0021     vname = varname;
0022     while ismember(vname, ws_vars)
0023         vname = sprintf('%s_%03d', varname, j);
0024         j = j + 1;
0025     end
0026 else
0027     vname = varname;
0028 end
0029
0030 disp(sprintf('** exporting to %s', vname));
0031
0032 % Get constructor string
0033 ch = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_constructor');
0034 cstr = get(ch, 'String');
0035
0036 cmd = sprintf('%s = %s;', vname, cstr);
0037 evalin('base', cmd);
0038
0039
0040
0041
0042

```


ltpda_constructor_helper_buildParams

PURPOSE ^

Build parameter panel on the main figure

SYNOPSIS ^

function ltpda_constructor_helper_buildParams(varargin)

DESCRIPTION ^

Build parameter panel on the main figure

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpda_constructor_helper_sets](#) Callback executed when user selects a parameter set

SOURCE CODE ^

```
0001 function ltpda_constructor_helper_buildParams(varargin)
0002 % Build parameter panel on the main figure
0003
0004
0005 mainfig = varargin{1};
0006 pl      = varargin{2};
0007
0008 % get selected class
0009 clh     = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
0010 cltxt   = get(clh, 'String');
0011 cl      = cltxt{get(clh, 'Value')};
0012
0013 % Convert plist to string
0014 str     = string(pl);
0015
0016 ch      = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_constructor');
0017
0018 set(ch, 'String', sprintf('%s(%s)', cl, str));
0019
0020
```


ltpda_constructor_helper_class

PURPOSE ^

Callback executed when user selects a class

SYNOPSIS ^

function ltpda_constructor_helper_class(varargin)

DESCRIPTION ^

Callback executed when user selects a class

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_constructor_helper_class(varargin)
0002 % Callback executed when user selects a class
0003
0004 mainfig = findobj('Tag', 'LTPDAconstructor_helper');
0005
0006 % get selected class
0007 clh = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
0008 cltxt = get(clh, 'String');
0009 cl = cltxt{get(clh, 'Value')};
0010
0011 lh = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_sets');
0012
0013
0014 % Get the info
0015 eval(sprintf('ii = %s.getInfo;', cl));
0016 sets = ii.sets;
0017 % eval(sprintf('sets = %s(%s, 'Params');', cl, cl));
0018
0019 % Fill set list
0020 if isa(sets, 'plist')
0021     setappdata(mainfig, 'plist', sets);
0022     set(lh, 'Value', 1);
0023     set(lh, 'String', {'Default'});
0024 else
0025     set(lh, 'Value', 1);
0026     set(lh, 'String', sets);
0027 end
0028
```


[Home](#) > [m](#) > [gui](#) > [constructor_helper](#) > ltpda_constructor_helper_sets.m

ltpda_constructor_helper_sets

PURPOSE ^

Callback executed when user selects a parameter set

SYNOPSIS ^

```
function ltpda_constructor_helper_sets(varargin)
```

DESCRIPTION ^

Callback executed when user selects a parameter set

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpda_constructor_helper_buildParams](#) Build parameter panel on the main figure

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_constructor_helper_sets(varargin)
0002
0003 % Callback executed when user selects a parameter set
0004
0005 mainfig = findobj('Tag', 'LTPDAconstructor_helper');
0006
0007 % get selected class
0008 clh = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
0009 cltxt = get(clh, 'String');
0010 cl = cltxt{get(clh, 'Value')};
0011
0012 % get selected set
0013 lh = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_sets');
0014 sets = get(lh, 'String');
0015 set = sets{get(lh, 'Value')};
0016
0017 if strcmp(set, 'Default')
0018     pl = getappdata(mainfig, 'plist');
0019 else
0020     cmd = sprintf('pl = %s.getInfo('%s', '%s').plist;', cl, cl, set);
0021     eval(cmd);
0022 end
0023
0024 ltpda_constructor_helper_buildParams(mainfig, pl);
0025
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

Contents

PURPOSE ^

%%%%%%%%%%%% path: m/gui/gltpda %%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%% path: m/gui/gltpda %%%%%%%%%%

    <a href="matlab:help m/gui/gltpda/buildLibrary">m/gui/gltpda/buildLibrary</a> - Function
to update the library with all the methods functions.
    <a href="matlab:help m/gui/gltpda/g_armmult">m/gui/gltpda/g_armmult</a> - This is the
automatic function wrapper
    <a href="matlab:help m/gui/gltpda/g_cleaninput">m/gui/gltpda/g_cleaninput</a> - This is
the automatic function wrapper
    <a href="matlab:help m/gui/gltpda/g_copyparams">m/gui/gltpda/g_copyparams</a> - This is
the automatic function wrapper
    <a href="matlab:help m/gui/gltpda/g_iplot">m/gui/gltpda/g_iplot</a> - This is the
automatic function wrapper
    <a href="matlab:help m/gui/gltpda/g_pl2ao">m/gui/gltpda/g_pl2ao</a> - For the block
plist2AO : let the user to build a plist and convert it
    <a href="matlab:help m/gui/gltpda/g_plot">m/gui/gltpda/g_plot</a> - This is the
automatic function wrapper
    <a href="matlab:help m/gui/gltpda/g_resp">m/gui/gltpda/g_resp</a> - $Id: g_resp.m,v
1.1 2008/03/01 13:43:20 nicola Exp $
    <a href="matlab:help m/gui/gltpda/g_saveresult">m/gui/gltpda/g_saveresult</a> - This is
the automatic function wrapper
    <a href="matlab:help m/gui/gltpda/loginLTPDA">m/gui/gltpda/loginLTPDA</a> - % LOGINDLG
Dialog for visually secure login.
    <a href="matlab:help m/gui/gltpda/ltpdagui">m/gui/gltpda/ltpdagui</a> -
=====
    <a href="matlab:help m/gui/gltpda/ltpdalib">m/gui/gltpda/ltpdalib</a> - Open LTPDA
Package Library.
    <a href="matlab:help m/gui/gltpda/ltpdasim">m/gui/gltpda/ltpdasim</a> - This is the
automatic function wrapper
    <a href="matlab:help m/gui/gltpda/pan1main">m/gui/gltpda/pan1main</a> - This is the
function for the main panel of the GUI LTPDA, the package of
    <a href="matlab:help m/gui/gltpda/pan2param">m/gui/gltpda/pan2param</a> - This is the
GUI m-file able to read the list of parameters to be set for
    <a href="matlab:help m/gui/gltpda/pan4import">m/gui/gltpda/pan4import</a> -
=====
    <a href="matlab:help m/gui/gltpda/pan5output">m/gui/gltpda/pan5output</a> -
=====
    <a href="matlab:help m/gui/gltpda/pan6globals">m/gui/gltpda/pan6globals</a> -
=====
    <a href="matlab:help m/gui/gltpda/pan7partial">m/gui/gltpda/pan7partial</a> -
=====
    <a href="matlab:help m/gui/gltpda/panXyettocome">m/gui/gltpda/panXyettocome</a> -
=====
    <a href="matlab:help m/gui/gltpda/progressBar">m/gui/gltpda/progressBar</a> - bar for
the LTPDA GUI.
    <a href="matlab:help m/gui/gltpda/slblocks">m/gui/gltpda/slblocks</a> - Defines the
block library for a specific Toolbox or Blockset.
    <a href="matlab:help m/gui/gltpda/uipickfiles">m/gui/gltpda/uipickfiles</a> -
uipickfiles: GUI program to select file(s) and/or directories.
    <a href="matlab:help m/gui/gltpda/welcome2LTPDA">m/gui/gltpda/welcome2LTPDA</a> -
Introductory window for the LTPDA Toolbox.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

[Home](#) > [m](#) > [gui](#) > [gltpda](#) > buildLibrary.m

buildLibrary

PURPOSE ^

Function to update the library with all the methods functions.

SYNOPSIS ^

```
function buildLibrary(varargin)
```

DESCRIPTION ^

Function to update the library with all the methods functions.
\$Id: buildLibrary.m,v 1.9 2008/08/04 20:07:33 nicola Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```

0001 function buildLibrary(varargin)
0002 % Function to update the library with all the methods functions.
0003 % $Id: buildLibrary.m,v 1.9 2008/08/04 20:07:33 nicola Exp $
0004
0005 % clear all
0006 bdclose('all');
0007 disp('=====')
0008 disp('Updating the LTPDA Library...');
0009
0010 % Retrieve the list of all functions and methods:
0011 ltpda_versions
0012
0013 % This creates a variable named 'versions', so structured:
0014 % | 1.filename | 2.functionName | 3.version | 4.date | 5.hour | 6.author | 7.folder |
0015 % | 8.hash | 9.category | 10.class |
0016 % [The 'class' is set to None if it's not a method]
0017
0017 origSize = size(versions,1); %#ok<NODEF>
0018 disp(['The overall number of LTPDA functions is ',num2str(origSize),'.'])
0019
0020
0021 % =====
0022 %% Remove functions not meant for the library:
0023 % =====
0024
0025 % Remove all test functions:
0026 for i=origSize:-1:1
0027     if numel(versions{i,2})>4 && strcmpi(versions{i,2}(1:5),'test_')
0028         versions(i,:)=[];
0029     end
0030 end
0031 newSize = size(versions,1);
0032 disp(['--- ',num2str(origSize-newSize), ' test functions ignored'])
0033 origSize = newSize;
0034
0035 % Remove all Internal functions:
0036 for i=origSize:-1:1
0037     if numel(versions{i,9})>7 && strcmpi(versions{i,9}(1:8),'Internal')
0038         versions(i,:)=[];
0039     end
0040 end
0041 newSize = size(versions,1);
0042 disp(['--- ',num2str(origSize-newSize), ' Internal functions ignored'])
0043 origSize = newSize;
0044

```

```

0045 % Remove all private functions:
0046 for i=origSize:-1:1
0047     x = numel(versions{i,10});
0048     if x>8 && strcmpi(versions{i,10}(x-7:x), '/private')
0049         versions(i,:)=[];
0050     end
0051 end
0052 newSize = size(versions,1);
0053 disp(['--- ',num2str(origSize-newSize), ' private functions ignored'])
0054 origSize = newSize;
0055
0056 % Remove all remaining examples:
0057 for i=origSize:-1:1
0058     x = numel(versions{i,7});
0059     if x>10 && strcmpi(versions{i,7}(x-9:x), '/examples/')
0060         versions(i,:)=[];
0061     end
0062 end
0063 newSize = size(versions,1);
0064 disp(['--- ',num2str(origSize-newSize), ' examples ignored'])
0065 origSize = newSize;
0066
0067 % Remove templates:
0068 for i=origSize:-1:1
0069     funcName = versions{i,2};
0070     x = numel(funcName);
0071     if x>9 && strcmpi(funcName(x-8:x), '_template')
0072         versions(i,:)=[];
0073     end
0074 end
0075 newSize = size(versions,1);
0076 disp(['--- ',num2str(origSize-newSize), ' templates ignored'])
0077 origSize = newSize;
0078
0079 % Remove all uncategorized functions:
0080 for i=origSize:-1:1
0081     if isempty(versions{i,9})
0082         versions(i,:)=[];
0083     end
0084 end
0085 newSize = size(versions,1);
0086 disp(['--- ',num2str(origSize-newSize), ' uncategorized functions ignored'])
0087 origSize = newSize;
0088
0089 % Remove all functions with no class:
0090 for i=origSize:-1:1
0091     if strcmp(versions{i,10}, 'None')
0092         versions(i,:)=[];
0093     end
0094 end
0095 newSize = size(versions,1);
0096 disp(['--- ',num2str(origSize-newSize), ' functions with class ''None'' ignored'])
0097 origSize = newSize;
0098
0099 % Remove all functions with non user classes:
0100 for i=origSize:-1:1
0101     if numel(versions{i,10})>5 && strncmpi(versions{i,10}, 'ltpda_', 6)
0102         versions(i,:)=[];
0103     elseif numel(versions{i,10})>3 && strncmpi(versions{i,10}, 'data', 4)
0104         versions(i,:)=[];
0105     end
0106 end
0107 newSize = size(versions,1);
0108 disp(['--- ',num2str(origSize-newSize), ' functions with class ''ltpda_****'
ignored'])
0109 origSize = newSize;
0110 % =====
0111 % Prepare the library for update:
0112 % =====
0113
0114 disp(['The number of LTPDA functions for the library is ',num2str(origSize),'.'])
0115 numbFunctions = 0;
0116 funcCategories = {};
0117 for i=1:origSize
0118     if numel(versions{i,10})>3 && strcmpi(versions{i,10}(1:4), 'None')
0119         numbFunctions = numbFunctions +1;
0120         if ~ismember(versions{i,9}, funcCategories)
0121             funcCategories = [funcCategories, versions{i,9}];
0122         end
0123     end
0124 end
0125 % These are the root level subsystems:
0126 funcCategories = [funcCategories, 'Methods'];
0127 disp([num2str(numbFunctions), ' are functions, ', num2str(origSize-numbFunctions), '
are methods.'])
0128 disp('=====')

```

```

0129
0130     load_system libraryBase.mdl
0131     open_ltpda_library.mdl
0132     set(get_param(gcs, 'Handle'), 'Lock', 'off')
0133
0134     % Delete all functions/methods folders in the root and create them anew:
0135     for i=1:numel(funcCategories)
0136         try %#ok<ALIGN>
0137             subsystem = find_system(bdroot, 'Name', funcCategories{i});
0138             subsysPos = get(get_param(subsystem{1}, 'Handle'), 'Position');
0139             delete_block(subsystem);
0140             catch end %#ok<CTCH>
0141             SubSystemBlock =
add_block('libraryBase/SubSystName', ['ltpda_library/', funcCategories{i}]);
0142             disp(['-- ', funcCategories{i}, ' updated'])
0143             if ~isempty(subsystem), set(SubSystemBlock, 'Position', subsysPos); end
0144             set(SubSystemBlock, 'LinkStatus', 'inactive')
0145             set(SubSystemBlock, 'MaskDisplay', ['disp(''', funcCategories{i}, ''')'])
0146             clear subsysPos subsystem SubSystemBlock
0147         end
0148
0149     % Creating the Methods/<class> folder for each class anew:
0150     classes = utils.helper.ltpda_classes;
0151
0152     % Remove improper classes:
0153     for i=numel(classes):-1:1
0154         if (numel(classes{i})>5 && strncmpi(classes{i}, 'ltpda_', 6)) ||
(numel(classes{i})>3 && strncmpi(classes{i}, 'data', 4)),
0155             classes(i)=[];
0156         end;
0157     end
0158
0159     for i=1:numel(classes)
0160         classFolder =
add_block('libraryBase/SubSystName', ['ltpda_library/Methods/', classes{i}]);
0161         set(classFolder, 'LinkStatus', 'inactive')
0162         set(classFolder, 'MaskDisplay', ['disp(''', classes{i}, ''')'])
0163         % Cycle to set proper position:
0164         x=1;y=1;
0165         while
~isempty(find_system('ltpda_library/Methods', 'SearchDepth', 1, 'Position', [30+80*(y-1) ,
30+80*(x-1) , 80+80*(y-1) , 80+80*(x-1)]))
0166             y=y+1;
0167             if y==6, x=x+1; y=1; end
0168         end
0169         set(classFolder, 'Position', [30+80*(y-1) , 30+80*(x-1) , 80+80*(y-1) , 80+80*(x-
1)])
0170     end
0171
0172     % =====
0173     %%                               Rebuild the library contents:
0174     % =====
0175
0176     for i=1:origSize
0177         currMethod = versions{i,2};
0178         currCateg = versions{i,9};
0179         currClass = versions{i,10};
0180
0181         % Check to verify whether a folder for this category exists:
0182         categFolder =
find_system(['ltpda_library/Methods/', currClass], 'SearchDepth', 1, 'Name', currCateg);
0183         try categFolder = categFolder{1}; catch end %#ok<CTCH>
0184         if isempty(categFolder)
0185             categFolder =
add_block('libraryBase/SubSystName', ['ltpda_library/Methods/', currClass, '/', currCateg]);
0186             set(categFolder, 'LinkStatus', 'inactive')
0187             set(categFolder, 'MaskDisplay', ['disp(''', currCateg, ''')'])
0188             % Cycle to set proper position:
0189             x=1;y=1;
0190             while
~isempty(find_system(['ltpda_library/Methods/', currClass], 'SearchDepth', 1, 'Position'
, [30+80*(y-1) , 30+80*(x-1) , 80+80*(y-1) , 80+80*(x-1)]))
0191                 aaa =
find_system(['ltpda_library/Methods/', currClass], 'SearchDepth', 1, 'Position', [30+80*(y-1) ,
30+80*(x-1) , 80+80*(y-1) , 80+80*(x-1)]);
0192                 if (strcmp(aaa{1}, ['ltpda_library/Methods/', currClass]) &&
numel(aaa)==1), break; end
0193                 y=y+1;
0194                 if y==5, x=x+1; y=1; end
0195             end
0196             set(categFolder, 'Position', [30+80*(y-1) , 30+80*(x-1) , 80+80*(y-1) ,
80+80*(x-1)])
0197         end
0198
0199         % Adding the proper method block:
0200         if strcmp(currCateg, 'Constructor')

```

```

0201         blockH =
add_block('libraryBase/constructor',[ 'ltpda_library/Methods/',currClass,'/',currCateg,'/'
,currMethod]);
0202         elseif strcmp(currCateg,'Output')
0203             blockH =
add_block('libraryBase/output',[ 'ltpda_library/Methods/',currClass,'/',currCateg,'/'
,currMethod]);
0204         elseif ~strcmp(currMethod,'Contents')
0205             blockH =
add_block('libraryBase/function',[ 'ltpda_library/Methods/',currClass,'/',currCateg,'/'
,currMethod]);
0206         end
0207
0208         % Cycle to set proper position:
0209         x=1;y=1;
0210         while
~isempty(find_system([ 'ltpda_library/Methods/',currClass,'/',currCateg], 'SearchDepth',1, 'Position'
,[30+70*(y-1) , 30+70*(x-1) , 80+70*(y-1) , 60+70*(x-1)]))
0211             aaa =
find_system([ 'ltpda_library/Methods/',currClass,'/',currCateg], 'SearchDepth',1, 'Position'
,[30+70*(y-1) , 30+70*(x-1) , 80+70*(y-1) , 60+70*(x-1)]);
0212             if (strcmp(aaa{1},[ 'ltpda_library/Methods/',currClass,'/',currCateg])) &&
numel(aaa)==1), break; end
0213             y=y+1;
0214             if y==5, x=x+1; y=1; end
0215         end
0216         set(blockH,'Position',[30+70*(y-1) , 30+70*(x-1) , 80+70*(y-1) , 60+70*(x-
1)])
0217
0218         set(blockH,'LinkStatus','inactive')
0219         set(blockH,'MaskDisplay',[ 'disp('',currMethod,'')'])
0220         set(blockH,'Tag',[ 'method ',currClass])
0221         set(blockH,'Description','')
0222         maskText = eval([ 'help('',currClass,'/',currMethod,'')']);
0223         maskText = strtok(maskText,'%');
0224         set(blockH,'MaskDescription',maskText);
0225         set(blockH,'OpenFcn','ltpdagui(''callback'',1,'redraw'',2);if
(~strcmp(bdroot,'ltpda_library') && ~isempty(findobj('Tag','LTPDAGUI'))),
figure(findobj('Tag','LTPDAGUI')); end');
0226         childpath = find_system(blockH,'LookUnderMasks','all','BlockType','M-S-
Function');
0227         set(childpath,'Tag',currMethod)
0228         set(childpath,'Name',currMethod)
0229         set(childpath,'Description','')
0230     end
0231
0232     % Setting proper position and size for subsystem folders in the root:
0233     subsystems =
find_system(bdroot,'LookUnderMasks','all','SearchDepth',1,'BlockType','SubSystem');
0234     x=1; y=1;
0235     for i=1:numel(subsystems)
0236         set_param(subsystems{i},'Position',[30+100*(y-1) , 30+100*(x-1) , 95+100*(y-1) ,
90+100*(x-1)])
0237         y=y+1;
0238         if y==4, x=x+1; y=1; end
0239     end
0240
0241     close_system libraryBase.mdl
0242
0243     % Update the annotation:
0244     annotationHandle =
find_system(bdroot,'FindAll','on','SearchDepth',1,'type','annotation');
0245     caption1 = 'LTPDA Library';
0246     caption2 = ['v3.0 - ',datestr(now,1)];
0247     caption = strvcats(caption1,caption2); %#ok<VCAT>
0248     set_param(annotationHandle,'HorizontalAlignment','center','Text',caption)
0249
0250     warning off all
0251     save_system ltpda_library.mdl
0252     warning on all
0253     close_system ltpda_library.mdl
0254
0255 end
0256

```

[Home](#) > [m](#) > [gui](#) > [gltpda](#) > [g_armmult.m](#)

g_armmult

PURPOSE [^]

This is the automatic function wrapper

SYNOPSIS [^]

```
function g_armmult(block)
```

DESCRIPTION [^]

```

    This is the automatic function wrapper
=====
===== level-2 M file S-function =====
=====
This wrapper is the automatic function, called by every function block
in Simulink, able to execute m-file functions retrieving from Simulink:
(1) the pointer to the AO(s) to be analyzed, coming in as input of the
    corresponding block (ie, the DATA),
(2) the name of the function to be applied on those data, from the tag
    of the currently executed block (ie, the true FUNCTION),
(3) the parameters for that particular block, retrieved from the global
    shared workspace by the handle of the block (ie, the PARAMETERS).

The output is then generated as:
    OUTPUT = FUNCTION(DATA,PARAMETERS)

This output in the end is saved into the global array containing all
the AOs (ie, all the DATA go together with other data): thus this output
will be freely accessible by all the other functions.

The only real output to Simulink will be just the ordinal number of the
so-generated AO into the global array of AOs.

=====
                For UNIVARIATE analysis functions
=====
(the difference with the other function, 'ltpdasimmulti', is the
dimension of the output port, here set = DimInputPorts)

$Id: g_armmult.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

-  [function setup\(block\)](#)
-  [function SetInpPortFrameData\(block, idx, fd\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE [^]

```

0001 function g_armmult(block)
0002
0003 %           This is the automatic function wrapper
0004 % =====
```



```

0005 % ===== level-2 M file S-function =====
0006 % =====
0007 % This wrapper is the automatic function, called by every function block
0008 % in Simulink, able to execute m-file functions retrieving from Simulink:
0009 % (1) the pointer to the AO(s) to be analyzed, coming in as input of the
0010 %     corresponding block (ie, the DATA),
0011 % (2) the name of the function to be applied on those data, from the tag
0012 %     of the currently executed block (ie, the true FUNCTION),
0013 % (3) the parameters for that particular block, retrieved from the global
0014 %     shared workspace by the handle of the block (ie, the PARAMETERS).
0015 %
0016 % The output is then generated as:
0017 %     OUTPUT = FUNCTION(DATA,PARAMETERS)
0018 %
0019 % This output in the end is saved into the global array containing all
0020 % the AOs (ie, all the DATA go together with other data): thus this output
0021 % will be freely accessible by all the other functions.
0022 %
0023 % The only real output to Simulink will be just the ordinal number of the
0024 % so-generated AO into the global array of AOs.
0025 %
0026 % =====
0027 %             For UNIVARIATE analysis functions
0028 % =====
0029 % (the difference with the other function, 'ltpdasimmulti', is the
0030 % dimension of the output port, here set = DimInputPorts)
0031 %
0032 %
0033 % $Id: g_armmult.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0034
0035 setup(block);
0036
0037 %%
0038 function setup(block)
0039 % global LTPDAinvar loopstatus
0040
0041
0042 %% Register dialog parameter: none, because they're retrieved directly
0043 %% from the memory. This will prevent the user to modify the parameters
0044 %% outside the proper parameters panel:
0045 block.NumDialogPrms = 0;
0046
0047 %% Register number of input and output ports
0048 block.NumInputPorts = 1;
0049 block.NumOutputPorts = 1;
0050
0051 %% Setup functional port properties to dynamically inherited.
0052 block.SetPreCompInpPortInfoToDynamic;
0053 block.SetPreCompOutPortInfoToDynamic;
0054
0055 block.InputPort(1).DirectFeedthrough = true;
0056 block.InputPort(1).DatatypeID = 0;
0057 block.InputPort(1).Complexity = 0;
0058 block.OutputPort(1).DatatypeID = 0;
0059 block.OutputPort(1).Complexity = 0;
0060 block.OutputPort(1).Dimensions = 1;
0061 %(block.InputPort(1).Dimensions)^2;
0062 block.SampleTimes = [0 0];
0063 block.SetAccelRunOnTLC(false);
0064
0065 %% Register methods
0066 block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
0067 block.RegBlockMethod('SetInputPortDimensions',@SetInpPortDims);
0068 % block.RegBlockMethod('SetOutputPortDimensions',@SetOutPortDims);
0069 block.RegBlockMethod('Outputs',@Outputs);
0070
0071 function SetInpPortFrameData(block, idx, fd)
0072 block.InputPort(1).SamplingMode = fd;
0073 block.OutputPort(1).SamplingMode = fd;
0074
0075 function SetInpPortDims(block, idx, di)
0076 block.InputPort(idx).Dimensions = di;
0077 % outdims=(block.InputPort(1).Dimensions)^2;
0078 % block.OutputPort(1).Dimensions = outdims;
0079
0080 % function SetOutPortDims(block, idx, di)
0081 % block.OutputPort(idx).Dimensions = 1;
0082
0083
0084 %%
0085 function Outputs(block)
0086 global LTPDAinvar
0087
0088 if length(block.InputPort(1).Data)~=2
0089     block.OutputPort(1).Data = block.InputPort(1).Data;
0090     disp('=====')

```

```
0091     disp('Something's wrong. Please check ltpdaarmult.m function')
0092     disp('=====')
0093     return
0094 end
0095
0096     outdata = LTPDAinvar{block.InputPort(1).Data(1)} .*
LTPDAinvar{block.InputPort(1).Data(2)};
0097     outdata = num2cell(outdata);
0098
0099     lineHandles = get_param(get_param(gcbbh, 'Parent'), 'LineHandles');
0100     signalName = get(lineHandles.Outputport, 'Name');
0101     if ~isempty(signalName) && numel(outdata)==1
0102         outdata{1} = setnh(outdata{1}, 'name', signalName);
0103     end
0104     xx = size(LTPDAinvar,1);
0105     LTPDAinvar(xx+1,:) = [outdata,0];
0106     block.OutputPort(1).Data = xx+1;
0107
0108 %endfunction
```

g_cleaninput

PURPOSE ^

This is the automatic function wrapper

SYNOPSIS ^

function g_cleaninput(block)

DESCRIPTION ^

```

    This is the automatic function wrapper
=====
===== level-2 M file S-function =====
=====
To remove an object from the list of the inputs.

$Id: g_cleaninput.m,v 1.1 2008/04/10 15:27:52 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function setup\(block\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function SetOutPortDims\(block, idx, di\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE ^

```

0001 function g_cleaninput(block)
0002
0003 %           This is the automatic function wrapper
0004 % =====
0005 % ===== level-2 M file S-function =====
0006 % =====
0007 % To remove an object from the list of the inputs.
0008 %
0009 % $Id: g_cleaninput.m,v 1.1 2008/04/10 15:27:52 nicola Exp $
0010
0011 setup(block);
0012
0013 %%
0014 function setup(block)
0015
0016 %% Register dialog parameter: none, because they're retrieved directly
0017 %% from the memory. This will prevent the user to modify the parameters
0018 %% outside the proper parameters panel:
0019 block.NumDialogPrms = 0;
0020
0021 %% Register number of input and output ports
0022 block.NumInputPorts = 2;
0023 block.NumOutputPorts = 1;
0024
0025 %% Setup functional port properties to dynamically inherited.
0026 block.SetPreCompInpPortInfoToDynamic;
0027 block.SetPreCompOutPortInfoToDynamic;
0028
```

```

0029 block.InputPort(1).DirectFeedthrough = true;
0030 block.InputPort(1).DatatypeID = 0;
0031 block.InputPort(1).Complexity = 0;
0032 % block.InputPort(1).Dimensions = 2;
0033 block.InputPort(2).DirectFeedthrough = true;
0034 block.InputPort(2).DatatypeID = 0;
0035 block.InputPort(2).Complexity = 0;
0036 % block.InputPort(2).Dimensions = 2;
0037 block.OutputPort(1).DatatypeID = 0;
0038 block.OutputPort(1).Complexity = 0;
0039 % block.OutputPort(1).Dimensions = 1;
0040 block.SampleTimes = [0 0];
0041 block.SetAccelerateRunOnTLC(false);
0042
0043 %% Register methods
0044 % block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
0045 block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
0046 block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0047 block.RegBlockMethod('Outputs', @Outputs);
0048
0049 % function SetInpPortFrameData(block, idx, fd)
0050 % block.InputPort(1).SamplingMode = fd;
0051 % block.OutputPort(1).SamplingMode = fd;
0052
0053 function SetInpPortDims(block, idx, di)
0054 block.InputPort(idx).Dimensions = di;
0055
0056 function SetOutPortDims(block, idx, di)
0057 block.OutputPort(idx).Dimensions = di;
0058
0059 %%
0060 function Outputs(block)
0061 global LTPDAinvar
0062
0063 LTPDAinvar{block.InputPort(2).Data,1} = [];
0064 block.OutputPort(1).Data = block.InputPort(1).Data;
0065
0066 %endfunction
0067

```

[Home](#) > [m](#) > [gui](#) > [gltpda](#) > [g_copyparams.m](#)

g_copyparams

PURPOSE [^]

This is the automatic function wrapper

SYNOPSIS [^]

function g_copyparams(block)

DESCRIPTION [^]

```

        This is the automatic function wrapper
=====
===== level-2 M file S-function =====
=====
To save as standalone variable(s) into global shared workspace the A0(s)
received as input from the corresponding block, executed in Simulink.

$Id: g_copyparams.m,v 1.1 2008/03/01 13:43:20 nicola Exp $

```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

-  [function setup\(block\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function SetOutPortDims\(block, idx, di\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE [^]

```

0001 function g_copyparams(block)
0002
0003 %           This is the automatic function wrapper
0004 % =====
0005 % ===== level-2 M file S-function =====
0006 % =====
0007 % To save as standalone variable(s) into global shared workspace the A0(s)
0008 % received as input from the corresponding block, executed in Simulink.
0009 %
0010 % $Id: g_copyparams.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0011
0012 setup(block);
0013
0014 %%
0015 function setup(block)
0016
0017 %% Register dialog parameter: none, because they're retrieved directly
0018 %% from the memory. This will prevent the user to modify the parameters
0019 %% outside the proper parameters panel:
0020 block.NumDialogPrms = 0;
0021
0022 %% Register number of input and output ports
0023 block.NumInputPorts = 2;
0024 block.NumOutputPorts = 1;
0025
0026 %% Setup functional port properties to dynamically inherited.
0027 block.SetPreCompInpPortInfoToDynamic;

```

```

0028     block.SetPreCompOutPortInfoToDynamic;
0029
0030     block.InputPort(1).DirectFeedthrough = true;
0031     block.InputPort(1).DatatypeID = 0;
0032     block.InputPort(1).Complexity = 0;
0033 %     block.InputPort(1).Dimensions = 2;
0034     block.InputPort(2).DirectFeedthrough = true;
0035     block.InputPort(2).DatatypeID = 0;
0036     block.InputPort(2).Complexity = 0;
0037 %     block.InputPort(2).Dimensions = 2;
0038     block.OutputPort(1).DatatypeID = 0;
0039     block.OutputPort(1).Complexity = 0;
0040 %     block.OutputPort(1).Dimensions = 1;
0041     block.SampleTimes = [0 0];
0042     block.SetAccelerateRunOnTLC(false);
0043
0044     %% Register methods
0045 %     block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
0046     block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
0047     block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0048     block.RegBlockMethod('Outputs', @Outputs);
0049
0050 %     function SetInpPortFrameData(block, idx, fd)
0051 %     block.InputPort(1).SamplingMode = fd;
0052 %     block.OutputPort(1).SamplingMode = fd;
0053
0054     function SetInpPortDims(block, idx, di)
0055     block.InputPort(idx).Dimensions = di;
0056
0057     function SetOutPortDims(block, idx, di)
0058     block.OutputPort(idx).Dimensions = di;
0059
0060 %%
0061 function Outputs(block)
0062
0063 % currBlk=get(gcblk,'Parent');
0064 currBlkConnect=get_param(get(gcblk,'Parent'),'PortConnectivity');
0065 parentBlkHandle=currBlkConnect(2).SrcBlock;
0066 childBlkHandle=currBlkConnect(3).DstBlock;
0067 param2copy=get_param(parentBlkHandle,'Description');
0068 set_param(childBlkHandle,'Description',param2copy);
0069 annotation=get_param(parentBlkHandle,'AttributesFormatString');
0070 set_param(childBlkHandle,'AttributesFormatString',annotation);
0071
0072     block.OutputPort(1).Data = block.InputPort(1).Data;
0073
0074 %endfunction
0075

```

g_iplot

PURPOSE ^

This is the automatic function wrapper

SYNOPSIS ^

function g_iplot(block)

DESCRIPTION ^

```

    This is the automatic function wrapper
=====
===== level-2 M file S-function =====
=====
To save as standalone variable(s) into global shared workspace the AO(s)
received as input from the corresponding block, executed in Simulink.

$Id: g_iplot.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function setup\(block\)](#)
-  [function SetInpPortFrameData\(block, idx, fd\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function SetOutPortDims\(block, idx, di\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE ^

```

0001 function g_iplot(block)
0002
0003 %           This is the automatic function wrapper
0004 % =====
0005 % ===== level-2 M file S-function =====
0006 % =====
0007 % To save as standalone variable(s) into global shared workspace the AO(s)
0008 % received as input from the corresponding block, executed in Simulink.
0009 %
0010 % $Id: g_iplot.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0011
0012 setup(block);
0013
0014 %%
0015 function setup(block)
0016 global LTPDAinvar;
0017
0018 %% Register dialog parameter: none, because they're retrieved directly
0019 %% from the memory. This will prevent the user to modify the parameters
0020 %% outside the proper parameters panel:
0021 block.NumDialogPrms = 0;
0022
0023 %% Register number of input and output ports
0024 block.NumInputPorts = 1;
0025 block.NumOutputPorts = 1;
0026
```

```

0027 %% Setup functional port properties to dynamically inherited.
0028 block.SetPreCompInpPortInfoToDynamic;
0029 block.SetPreCompOutPortInfoToDynamic;
0030
0031 block.InputPort(1).DirectFeedthrough = true;
0032 block.InputPort(1).DatatypeID = 0;
0033 block.InputPort(1).Complexity = 0;
0034 % block.InputPort(1).Dimensions = 2;
0035 block.OutputPort(1).DatatypeID = 0;
0036 block.OutputPort(1).Complexity = 0;
0037 % block.OutputPort(1).Dimensions = 1;
0038 block.SampleTimes = [0 0];
0039 block.SetAccelerateRunOnTLC(false);
0040
0041 %% Register methods
0042 block.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);
0043 block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
0044 block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0045 block.RegBlockMethod('Outputs', @Outputs);
0046
0047 function SetInpPortFrameData(block, idx, fd)
0048 block.InputPort(1).SamplingMode = fd;
0049 block.OutputPort(1).SamplingMode = fd;
0050
0051 function SetInpPortDims(block, idx, di)
0052 block.InputPort(idx).Dimensions = di;
0053
0054 function SetOutPortDims(block, idx, di)
0055 block.OutputPort(idx).Dimensions = di;
0056
0057
0058 %%
0059 function Outputs(block)
0060 global LTPDAinvar
0061
0062 % figure
0063 hold on
0064 switch length(block.InputPort(1).Data)
0065     case 1
0066         iplot(LTPDAinvar{block.InputPort(1).Data});
0067     case 2
0068         iplot(LTPDAinvar{block.InputPort(1).Data(1)}, LTPDAinvar{block.InputPort(1).Data(2)});
0069     case 3
0070         iplot(LTPDAinvar{block.InputPort(1).Data(1)}, LTPDAinvar{block.InputPort(1).Data(2)}, LTPDAinvar{block.InputPort(1).Data(3)});
0071     case 4
0072         iplot(LTPDAinvar{block.InputPort(1).Data(1)}, LTPDAinvar{block.InputPort(1).Data(2)}, LTPDAinvar{block.InputPort(1).Data(3)}, LTPDAinvar{block.InputPort(1).Data(4)});
0073 end
0074
0075 % %=====
0076
0077 block.OutputPort(1).Data = block.InputPort(1).Data;
0078
0079 %endfunction
0080

```


g_pl2ao

PURPOSE ^

For the block plist2AO : let the user to build a plist and convert it

SYNOPSIS ^

function g_pl2ao(block)

DESCRIPTION ^

For the block plist2AO : let the user to build a plist and convert it into an AO in memory at the time of execution.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function setup\(block\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE ^

```
0001 function g_pl2ao(block)
0002 % For the block plist2AO : let the user to build a plist and convert it
0003 % into an AO in memory at the time of execution.
0004
0005 % $Id: g_pl2ao.m,v 1.1 2008/03/01 13:43:20 nicola Exp $Revision: 1.1 $
0006
0007 setup(block);
0008
0009 %%
0010 function setup(block)
0011
0012     %% Register dialog parameter: none, because they're retrieved directly
0013     %% from the memory. This will prevent the user to modify the parameters
0014     %% outside the proper parameters panel:
0015     block.NumDialogPrms = 0;
0016
0017     %% Register number of input and output ports
0018     block.NumInputPorts = 0;
0019     block.NumOutputPorts = 1;
0020
0021     %% Setup functional port properties to dynamically inherited.
0022     block.SetPreCompOutPortInfoToDynamic;
0023
0024     block.OutputPort(1).DatatypeID = 0;
0025     block.OutputPort(1).Complexity = 0;
0026     block.OutputPort(1).Dimensions = 1;
0027     block.OutputPort(1).SamplingMode = 'sample';
0028     block.SampleTimes = [0 0];
0029     block.SetAccelerRunOnTLC(false);
0030
0031     %% Register methods
0032     % block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0033     block.RegBlockMethod('Outputs', @Outputs);
0034
0035     % function SetOutPortDims(block, idx, di)
0036     % block.OutputPort(idx).Dimensions = 1;
0037
```

```
0038
0039 %%
0040 function Outputs(block)
0041 global LTPDAinvar
0042
0043 currhandle=gcbh;
0044 currparent=get_param(currhandle, 'Parent');
0045
0046 paramcommand = get_param(currparent, 'Description');
0047 eval(paramcommand)
0048 try
0049     outdata = ao(params);
0050 catch
0051     outdata = ao(1);
0052 end
0053
0054
0055 xx = size(LTPDAinvar,1);
0056 LTPDAinvar(xx+1,:) = [num2cell(outdata),0];
0057 % LTPDAinvar{xx+1} = {outdata,0};
0058 block.OutputPort(1).Data = xx+1;
0059
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

g_plot

PURPOSE ^

This is the automatic function wrapper

SYNOPSIS ^

function g_plot(block)

DESCRIPTION ^

```

    This is the automatic function wrapper
=====
===== level-2 M file S-function =====
=====
To save as standalone variable(s) into global shared workspace the AO(s)
received as input from the corresponding block, executed in Simulink.

$Id: g_plot.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function setup\(block\)](#)
-  [function SetInpPortFrameData\(block, idx, fd\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function SetOutPortDims\(block, idx, di\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE ^

```

0001 function g_plot(block)
0002
0003 %           This is the automatic function wrapper
0004 % =====
0005 % ===== level-2 M file S-function =====
0006 % =====
0007 % To save as standalone variable(s) into global shared workspace the AO(s)
0008 % received as input from the corresponding block, executed in Simulink.
0009 %
0010 % $Id: g_plot.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0011
0012 setup(block);
0013
0014 %%
0015 function setup(block)
0016 global AnOb;
0017
0018 %% Register dialog parameter: none, because they're retrieved directly
0019 %% from the memory. This will prevent the user to modify the parameters
0020 %% outside the proper parameters panel:
0021 block.NumDialogPrms = 0;
0022
0023 %% Register number of input and output ports
0024 block.NumInputPorts = 1;
0025 block.NumOutputPorts = 1;
0026
```

```

0027 %% Setup functional port properties to dynamically inherited.
0028 block.SetPreCompInpPortInfoToDynamic;
0029 block.SetPreCompOutPortInfoToDynamic;
0030
0031 block.InputPort(1).DirectFeedthrough = true;
0032 block.InputPort(1).DatatypeID = 0;
0033 block.InputPort(1).Complexity = 0;
0034 % block.InputPort(1).Dimensions = 2;
0035 block.OutputPort(1).DatatypeID = 0;
0036 block.OutputPort(1).Complexity = 0;
0037 % block.OutputPort(1).Dimensions = 1;
0038 block.SampleTimes = [0 0];
0039 block.SetAccelerateRunOnTLC(false);
0040
0041 %% Register methods
0042 block.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);
0043 block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
0044 block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0045 block.RegBlockMethod('Outputs', @Outputs);
0046
0047 function SetInpPortFrameData(block, idx, fd)
0048 block.InputPort(1).SamplingMode = fd;
0049 block.OutputPort(1).SamplingMode = fd;
0050
0051 function SetInpPortDims(block, idx, di)
0052 block.InputPort(idx).Dimensions = di;
0053
0054 function SetOutPortDims(block, idx, di)
0055 block.OutputPort(idx).Dimensions = di;
0056
0057
0058 %%
0059 function Outputs(block)
0060 global AnOb
0061
0062 y=length(block.InputPort(1).Data);
0063 figure
0064 hold on
0065 for j=1:y
0066     plot(AnOb{block.InputPort(1).Data(j)})
0067 end
0068
0069 % %=====
0070
0071     block.OutputPort(1).Data = block.InputPort(1).Data;
0072
0073 %endfunction
0074

```

[Home](#) > [m](#) > [gui](#) > [gltpda](#) > [g_resp.m](#)

g_resp

PURPOSE ^

\$Id: g_resp.m,v 1.1 2008/03/01 13:43:20 nicola Exp \$

SYNOPSIS ^

```
function g_resp(block)
```

DESCRIPTION ^

\$Id: g_resp.m,v 1.1 2008/03/01 13:43:20 nicola Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  [function setup\(block\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function SetInpPortFrameData\(block, idx, fd\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE ^

```

0001 function g_resp(block)
0002
0003 % $Id: g_resp.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0004
0005 setup(block);
0006
0007 %%
0008 function setup(block)
0009
0010 %% Register dialog parameter: none, because they're retrieved directly
0011 %% from the memory. This will prevent the user to modify the parameters
0012 %% outside the proper parameters panel:
0013 block.NumDialogPrms = 0;
0014
0015 %% Register number of input and output ports
0016 block.NumInputPorts = 1;
0017 block.NumOutputPorts = 1;
0018
0019 %% Setup functional port properties to dynamically inherited.
0020 block.SetPreCompOutPortInfoToDynamic;
0021
0022 block.InputPort(1).DirectFeedthrough = true;
0023 block.InputPort(1).DatatypeID = 0;
0024 block.InputPort(1).Complexity = 0;
0025 block.InputPort(1).SamplingMode = 'sample';
0026 block.OutputPort(1).DatatypeID = 0;
0027 block.OutputPort(1).Complexity = 0;
0028 block.OutputPort(1).Dimensions = 1;
0029 block.OutputPort(1).SamplingMode = 'sample';
0030 block.SampleTimes = [0 0];
0031 block.SetAcceIRunOnTLC(false);
0032
0033
0034

```

```

0035 %% Register methods
0036 block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
0037 block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
0038 % block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0039 block.RegBlockMethod('Outputs', @Outputs);
0040
0041 function SetInpPortDims(block, idx, di)
0042 block.InputPort(idx).Dimensions = di;
0043
0044 function SetInpPortFrameData(block, idx, fd)
0045 block.InputPort(1).SamplingMode = fd;
0046 block.OutputPort(1).SamplingMode = fd;
0047
0048 % function SetOutPortDims(block, idx, di)
0049 % block.OutputPort(idx).Dimensions = 1;
0050
0051 %%
0052 %%
0053 function Outputs(block)
0054 global LTPDAinvar
0055
0056
0057 currhandle=gcbh;
0058 currparent=get_param(currhandle, 'Parent');
0059
0060 paramcommand = get_param(currparent, 'Description');
0061 eval(paramcommand)
0062
0063
0064
0065 data = get(params.params(1), 'val');
0066 aodata = find(params, 'AO data');
0067 if ~isempty(aodata)
0068     currparam = LTPDAinvar{block.InputPort(1).Data,1}.data.x;
0069 end
0070
0071 outdata=resp(data,currparam);
0072 outdata = num2cell(outdata);
0073
0074 xx = size(LTPDAinvar,1);
0075 LTPDAinvar(xx+1,:) = [outdata,0];
0076 block.OutputPort(1).Data = xx+1;

```

[Home](#) > [m](#) > [gui](#) > [gltpda](#) > [g_saveresult.m](#)

g_saveresult

PURPOSE [^]

This is the automatic function wrapper

SYNOPSIS [^]

function g_saveresult(block)

DESCRIPTION [^]

```

                This is the automatic function wrapper
=====
===== level-2 M file S-function =====
=====
To save as standalone variable(s) into global shared workspace the AO(s)
received as input from the corresponding block, executed in Simulink.

$Id: g_saveresult.m,v 1.4 2008/03/21 16:13:54 nicola Exp $

```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

-  [function setup\(block\)](#)
-  [function SetInpPortFrameData\(block, idx, fd\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function SetOutPortDims\(block, idx, di\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE [^]

```

0001 function g_saveresult(block)
0002
0003 %                This is the automatic function wrapper
0004 % =====
0005 % ===== level-2 M file S-function =====
0006 % =====
0007 % To save as standalone variable(s) into global shared workspace the AO(s)
0008 % received as input from the corresponding block, executed in Simulink.
0009 %
0010 % $Id: g_saveresult.m,v 1.4 2008/03/21 16:13:54 nicola Exp $
0011
0012 setup(block);
0013
0014 %%
0015 function setup(block)
0016
0017 %% Register dialog parameter: none, because they're retrieved directly
0018 %% from the memory. This will prevent the user to modify the parameters
0019 %% outside the proper parameters panel:
0020 block.NumDialogPrms = 0;
0021
0022 %% Register number of input and output ports
0023 block.NumInputPorts = 1;
0024 block.NumOutputPorts = 1;
0025
0026 %% Setup functional port properties to dynamically inherited.

```

```

0027 block.SetPreCompInpPortInfoToDynamic;
0028 block.SetPreCompOutPortInfoToDynamic;
0029
0030 block.InputPort(1).DirectFeedthrough = true;
0031 block.InputPort(1).DatatypeID = 0;
0032 block.InputPort(1).Complexity = 0;
0033 % block.InputPort(1).Dimensions = 2;
0034 block.OutputPort(1).DatatypeID = 0;
0035 block.OutputPort(1).Complexity = 0;
0036 % block.OutputPort(1).Dimensions = 1;
0037 block.SampleTimes = [0 0];
0038 block.SetAccelerateRunOnTLC(false);
0039
0040 %% Register methods
0041 block.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);
0042 block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
0043 block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
0044 block.RegBlockMethod('Outputs', @Outputs);
0045
0046 function SetInpPortFrameData(block, idx, fd)
0047 block.InputPort(1).SamplingMode = fd;
0048 block.OutputPort(1).SamplingMode = fd;
0049
0050 function SetInpPortDims(block, idx, di)
0051 block.InputPort(idx).Dimensions = di;
0052
0053 function SetOutPortDims(block, idx, di)
0054 block.OutputPort(idx).Dimensions = di;
0055
0056
0057 %%
0058 function Outputs(block)
0059 global LTPDAinvar LTPDAoutvar
0060
0061 % Check if the user wants to stop the execution:
0062 lastChar = get(findobj('Name','LTPDA Progress Bar'),'CurrentCharacter');
0063 if ~isempty(lastChar) && strcmp(lastChar,'x')
0064     set_param(bdroot, 'SimulationCommand', 'stop')
0065     return
0066 end
0067
0068 y = numel(block.InputPort(1).Data);
0069 for j=1:y
0070     yy = numel(LTPDAinvar{block.InputPort(1).Data(j)});
0071     if yy > 1
0072         for jj=1:yy
0073             LTPDAoutvar = [LTPDAoutvar;{LTPDAinvar{block.InputPort(1).Data(j)}(jj)}];
0074         end
0075     else
0076         LTPDAoutvar = [LTPDAoutvar;{LTPDAinvar{block.InputPort(1).Data(j)}}];
0077     end
0078 end
0079
0080 % %=====
0081
0082 block.OutputPort(1).Data = block.InputPort(1).Data;
0083
0084 %endfunction
0085

```


loginLTPDA

PURPOSE ^

LOGINDLG Dialog for visually secure login.

SYNOPSIS ^

function [varargout]=loginLTPDA(varargin)

DESCRIPTION ^

```
LOGINDLG    Dialog for visually secure login.

Author:  Jeremy Smith
Date:    September 24, 2005
Last Edit:  July 7, 2006
Version:  1.2
Tested on: Matlab 7.0.4.365 (R14) Service Pack 2 and Matlab 7.1 SP 3
Description: custom login dialog because Matlab doesn't have an option
              for characters in an edit field to be replaced by asterixes
              (password security)

=====
=====  UPDATED FOR USE WITH LTPDA PACKAGE  =====
=====

The function now has a fixed supported syntax to call:
    [login password] = loginLTPDA('UserID','password')
or
    [login password] = loginLTPDA('','')

It will only accept 2 string inputs and it will provide 2 string
outputs.
If 2 empty strings are passed the edit fields will be empty, otherwise
they'll show as defaults the strings passed.

    $Id: loginLTPDA.m,v 1.6 2008/08/24 14:57:44 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [welcome2LTPDA](#) Introductory window for the LTPDA Toolbox.

This function is called by:

 [ltpdagui](#) =====
 [pan1main](#) This is the function for the main panel of the GUI LTPDA, the package of

SUBFUNCTIONS ^

-  [function KeyPress_Function\(h,eventdata,fig\)](#)
-  [function KeyPress_Func2\(h,eventdata,fig\)](#)
-  [function Cancel\(h,eventdata,fig\)](#)
-  [function OK\(h,eventdata,fig\)](#)
-  [function CancelReturn\(h,eventdata,fig\)](#)
-  [function OKReturn\(h,eventdata,fig\)](#)
-  [function Escape\(h,eventdata,fig\)](#)

SOURCE CODE ^

```

0001 function [varargout]=loginLTPDA(varargin)
0002 % LOGINDLG Dialog for visually secure login.
0003 %
0004 % Author: Jeremy Smith
0005 % Date: September 24, 2005
0006 % Last Edit: July 7, 2006
0007 % Version: 1.2
0008 % Tested on: Matlab 7.0.4.365 (R14) Service Pack 2 and Matlab 7.1 SP 3
0009 % Description: custom login dialog because Matlab doesn't have an option
0010 % for characters in an edit field to be replaced by asterixes
0011 % (password security)
0012 %
0013 % =====
0014 % ===== UPDATED FOR USE WITH LTPDA PACKAGE =====
0015 % =====
0016 %
0017 % The function now has a fixed supported syntax to call:
0018 % [login password] = loginLTPDA('userID','password')
0019 % or
0020 % [login password] = loginLTPDA('','')
0021 %
0022 % It will only accept 2 string inputs and it will provide 2 string
0023 % outputs.
0024 % If 2 empty strings are passed the edit fields will be empty, otherwise
0025 % they'll show as defaults the strings passed.
0026 %
0027 % $Id: loginLTPDA.m,v 1.6 2008/08/24 14:57:44 nicola Exp $
0028
0029 Login = varargin{1};
0030 password = varargin{2};
0031 for i=1: numel(password), passwordHidden(i)='*'; end
0032 if ~exist('passwordHidden','var'), passwordHidden = ''; end
0033
0034 set(0,'Units','pixels')
0035 Screen = get(0,'screensize');
0036 loginSize = [230 130];
0037 Position = [(Screen(3)-loginSize(1))/2 (Screen(4)-loginSize(2))/2 loginSize];
0038
0039
0040 % Create the GUI
0041 gui.main = dialog('HandleVisibility','on',...
0042 'Color','white',...
0043 'IntegerHandle','off',...
0044 'Menubar','none',...
0045 'Name','LTPDA Login',...
0046 'NumberTitle','off',...
0047 'Name','Login',...
0048 'Tag','logindlg',...
0049 'Units','pixels',...
0050 'Userdata','logindlg',...
0051 'Position',Position);
0052
0053 set(gui.main,'Closerequestfcn',{@Cancel,gui.main},'KeyPressfcn',{@Escape,gui.main})
0054
0055 % Texts
0056 gui.login_text = uicontrol(gui.main,'Style','text','BackgroundColor','white','FontSize',8,'HorizontalAlign','center','Units','pixels','String','Login','Position',[0 105
0057 Position(3) 20]);
0058 gui.password_text = uicontrol(gui.main,'Style','text','BackgroundColor','white','FontSize',8,'HorizontalAlign','center','Units','pixels','String','Password','Position',[0 60
0059 Position(3) 20]);
0060
0061 % Edits
0062 gui.edit1 =
0063 uicontrol(gui.main,'Tag','userid','Style','edit','FontSize',8,'HorizontalAlign','center','BackgroundColor','white','Units','pixels','String',Login,'KeyPressfcn',{@KeyPress_Func2,gui.main},
0064 [5 85 Position(3)-10 24],'Userdata',0);
0065 gui.edit2 =
0066 uicontrol(gui.main,'Tag','password','Style','edit','FontSize',8,'HorizontalAlign','center','BackgroundColor','white','Units','pixels','String',passwordHidden,'Position',[5 40
0067 Position(3)-10 24],'KeyPressfcn',{@KeyPress_Function,gui.main},'Userdata',password);
0068
0069 % Buttons
0070 buttonSize = [75 26];
0071 gui.OK = uicontrol(gui.main,'Style','push','FontSize',8,'Units','pixels','String','OK','Position',[Position(3)/2-buttonSize(1)-1 5
0072 buttonSize],'KeyPressFcn',{@OKReturn,gui.main},'Callback',{@OK,gui.main});
0073 gui.Cancel = uicontrol(gui.main,'Style','push','FontSize',8,'Units','pixels','String','Cancel','Position',[Position(3)/2+1 5
0074 buttonSize],'KeyPressFcn',{@CancelReturn,gui.main},'Callback',{@Cancel,gui.main});
0075
0076 % Images
0077 logosize = [31,31];
0078 dimension = [5 3 logosize];
0079 logo = axes('Parent',gui.main,'Units','pixels','Position',dimension);
0080 image(imread('buttonyes.jpg'),'Parent',logo);
0081 axis(logo,'off');
0082 logosize = [31,31];
0083 dimension = [Position(3)-logosize(1)-5 3 logosize];
0084 logo = axes('Parent',gui.main,'Units','pixels','Position',dimension);
0085 image(imread('buttonno.jpg'),'Parent',logo);
0086 axis(logo,'off');
0087
0088 setappdata(0,'logindlg',gui) % Save handle data
0089

```

```

0081 setappdata(gui.main, 'Check',0) % Error check setup. If Check remains 0 an empty cell array will be returned
0082
0083 uicontrol(gui.edit1) % Make the first edit box active
0084 % uicontrol(gui.OK) % Make the OK button active
0085
0086 % Pause the GUI and wait for a button to be pressed
0087 uiwait(gui.main)
0088
0089 try if varargin{3}, welcome2LTPDA(varargin{4},varargin{5}); pause(1); end; catch end
0090
0091 Check = getappdata(gui.main, 'Check'); % Check to see if a button was pressed
0092
0093 % Format output
0094 if Check == 1
0095     Login = get(gui.edit1, 'String');
0096     Password = get(gui.edit2, 'Userdata');
0097
0098     varargout(1) = {Login};
0099     varargout(2) = {Password};
0100 else % If OK wasn't pressed output empty fields
0101     varargout(1) = {' '};
0102     varargout(2) = {' '};
0103 end
0104
0105 delete(gui.main) % Close the GUI
0106 setappdata(0, 'logindlg', []) % Erase handles from memory
0107
0108 %% Hide Password
0109 function KeyPress_Function(h,eventdata,fig)
0110 % Function to replace all characters in the password edit box with
0111 % asterixes
0112 password = get(h, 'Userdata');
0113 key = get(fig, 'currentkey');
0114
0115 switch key
0116     case 'backspace'
0117         password = password(1:end-1); % Delete the last character in the password
0118     case 'return' % This cannot be done through callback without making tab to the same thing
0119         gui = getappdata(0, 'logindlg');
0120         OK([], [], gui.main);
0121     case 'tab' % Avoid tab triggering the OK button
0122         gui = getappdata(0, 'logindlg');
0123         uicontrol(gui.OK);
0124     otherwise
0125         password = [password get(fig, 'currentcharacter')]; % Add the typed character to the password
0126 end
0127
0128 SizePass = size(password); % Find the number of asterixes
0129 if SizePass(2) > 0
0130     asterix(1,1:SizePass(2)) = '*'; % Create a string of asterixes the same size as the password
0131     set(h, 'String', asterix) % Set the text in the password edit box to the asterix string
0132 else
0133     set(h, 'String', '')
0134 end
0135
0136 set(h, 'Userdata', password) % Store the password in its current state
0137
0138 function KeyPress_Func2(h,eventdata,fig)
0139 % To check whenever the user press Enter
0140 if ~isempty(get(fig, 'CurrentCharacter')) && real(get(fig, 'CurrentCharacter'))==13 && get(h, 'UserData')==0
0141     setappdata(fig, 'Check', 1)
0142     uiresume(fig)
0143 elseif real(get(fig, 'CurrentCharacter'))==13
0144     uicontrol(findobj(gcf, 'Tag', 'password'))
0145 else
0146     set(h, 'UserData', 1)
0147 end
0148
0149 %% Cancel
0150 function Cancel(h,eventdata,fig)
0151     uiresume(fig)
0152
0153 %% OK
0154 function OK(h,eventdata,fig)
0155 % Set the check and resume
0156     setappdata(fig, 'Check', 1)
0157     uiresume(fig)
0158
0159 function CancelReturn(h,eventdata,fig)
0160     if real(get(fig, 'CurrentCharacter'))==13
0161         uiresume(fig)
0162     end
0163
0164 %% OK
0165 function OKReturn(h,eventdata,fig)
0166 % Set the check and resume
0167     if real(get(fig, 'CurrentCharacter'))==13
0168         setappdata(fig, 'Check', 1)
0169         uiresume(fig)

```

```
0170     end
0171
0172     %% Escape
0173 function Escape(h,eventdata,fig)
0174 % Close the login if the escape button is pushed and neither edit box is
0175 % active
0176 key = get(fig,'currentkey');
0177
0178 if isempty(strfind(key,'escape')) == 0 && h == fig
0179     Cancel([],[],fig)
0180 end
```

ltpdagui

PURPOSE ^

```
=====
```

SYNOPSIS ^

```
function ltpdagui(varargin)
```

DESCRIPTION ^

```
=====
                                LTPDA GUI
=====

Open the main LTPDA GUI.

$Id: ltpdagui.m,v 1.21 2008/08/30 12:27:08 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [loginLTPDA](#) LOGINDLG Dialog for visually secure login.
-  [ltpdagui](#) =====
-  [pan1main](#) This is the function for the main panel of the GUI LTPDA, the package of
-  [pan2param](#) This is the GUI m-file able to read the list of parameters to be set for
-  [pan4import](#) =====
-  [pan5output](#) =====
-  [pan6globals](#) =====
-  [pan7partial](#) =====
-  [panXyettocome](#) =====

This function is called by:

-  [ltpdagui](#) =====
-  [pan1main](#) This is the function for the main panel of the GUI LTPDA, the package of
-  [pan7partial](#) =====

SUBFUNCTIONS ^

- function TopBottomTabCbK(hobj,evdt,indx,numbrow)
- function TopBottomTabResizeCbK(varargin)
- function GUIResize(varargin)
- function CheckTimerAlive(varargin)
- function HelpCallback(varargin)
- functionname = get_param(childpath,
- functionname = description(funcname+14:i-2);

SOURCE CODE ^

```

0001 function ltpdagui(varargin)
0002
0003 % =====
0004 %                               LTPDA GUI
0005 % =====
0006 %
0007 %   Open the main LTPDA GUI.
0008 %
0009 %   $Id: ltpdagui.m,v 1.21 2008/08/30 12:27:08 nicola Exp $
0010
0011 warning off all
0012 global LTPDAinvar LTPDAoutvar active labels scaleX
0013
0014 tabbedFigName = 'GUI Fawkes';
0015 screenSize = get(0,'screensize');
0016 scaleX     = screenSize(3)/1280;
0017 if scaleX<1, scaleX = 1; end
0018 setappdata(0,'ltpda_gui_scalefactor',scaleX);
0019 guiSize = [1 1 800*scaleX 550*scaleX];
0020 guiSize = round([(screenSize(3)-guiSize(3))/2 , (screenSize(4)-guiSize(4))/2 , guiSize(3) , guiSize(4)]);
0021
0022 if ~isempty(findobj('Name',tabbedFigName)) && nargin==0, return; end
0023
0024 if isempty(LTPDAinvar), LTPDAinvar = {}; end
0025 if isempty(LTPDAoutvar), LTPDAoutvar = {}; end
0026
0027 userID = getappdata(0,'ltpda_repo_user');
0028 if isempty(userID), userID = ''; end
0029 passwd = getappdata(0,'ltpda_repo_pass');
0030 if isempty(passwd), passwd = ''; end
0031
0032 if nargin==0, setappdata(0,'startup',1); [userID passwd] = loginLTPDA(userID,'',1,tabbedFigName,guiSize); try
0033 cd(getappdata(0,'DefaultDataDirectory')); catch end; end
0034 if isempty(userID), userID = 'JohnDoe'; passwd=''; end
0035
0036 setappdata(0,'ltpda_repo_user',userID)
0037 setappdata(0,'ltpda_repo_pass',passwd)
0038
0039 if nargin==0
0040     clc
0041     disp('Please wait: starting LTPDA GUI...')
0042     disp('Loading Simulink...')
0043     drawnow
0044     load_system('simulink')
0045     load_system('ltpda_library')
0046 end
0047
0048 try stop(timerfindall); delete(timerfindall); catch end
0049 t1 = timer('TimerFcn',@CheckTimerAlive,'ExecutionMode','fixedRate','Period',0.3);

```

```

0049 t2 = timer('TimerFcn',@CheckTimerAlive,'ExecutionMode','fixedRate','Period',1);
0050 start(t1)
0051 start(t2)
0052
0053 %%
0054
0055 tag = '';
0056 units = 'normalized';
0057 position = [0,0,1,1];
0058
0059 % text color in the label:
0060 foregroundcolor = [0.2,0.2,0.2];
0061 % color in the background, behind tabs:
0062 backgroundcolor = [0.7,0.7,1];
0063 % background color, inside every tab:
0064 panelbackgroundcolor = [0.98,0.98,0.98];
0065 % color of the small highlight line, above the selected tab
0066 titlehighlightcolor = [1,0.3,0];
0067
0068 horizontalalignment = 'center';
0069
0070 active = 0;
0071 string = {'Main LTPDA panel','Partial execution','Output';'Parameters','Nested loops','';'Repository','Import from
workspace','Globals'};
0072 nrows = size(string,1);
0073 if nrows>1
0074     for nn=1:nrows, labels{nn} = string(nn,:); end
0075     string = string(1,:);
0076 end
0077 ntab = numel(string);
0078 for jj=ntab:-1:1, ntab = ntab - strcmp(string(jj),''); end
0079
0080 % Cycle to retrieve parameters from the user:
0081 for mm=1:2:nargin
0082     property = lower(varargin{mm});
0083     value = varargin{mm+1};
0084     switch property
0085     case {'title','string'}
0086         string = value;
0087         nrows = size(string,1);
0088         if nrows>1
0089             for nn=1:nrows, labels{nn} = string(nn,:);end
0090             string = string(1,:);
0091         end
0092         ntab = numel(string);
0093         for jj=ntab:-1:1, ntab = ntab - strcmp(string(jj),''); end
0094     case {'frameposition','position'}
0095         position = value;
0096     case {'framebackgroundcolor','backgroundcolor'}
0097         backgroundcolor = value;
0098     case {'framebordertype','bordertype'}
0099         bordertype = value;
0100     case {'titleforegroundcolor','foregroundcolor'}
0101         foregroundcolor = value;
0102     case {'selecteditem','active'}
0103         active = value;
0104     case 'redraw'
0105         if value == 1 % main panel
0106             TopBottomTabCbK([],[],1,1)
0107             active = 1;
0108             return
0109         end
0110         if value == 1.1 % the user clicked the Start button
0111             TopBottomTabCbK([],[],1,1)
0112             active = 1;
0113             runSim = 1;
0114         end

```

```

0115         if value == 2 % Parameters panel
0116             TopBottomTabCbK([],[],1,2)
0117             active = 1;
0118             return
0119         end
0120         if value == 6 % globals panel
0121             TopBottomTabCbK([],[],3,3)
0122             active = 1;
0123             return
0124         end
0125         if value == 8 % nested loops
0126             TopBottomTabCbK([],[],2,2)
0127             active = 1;
0128             return
0129         end
0130
0131         case 'callback'
0132             if value==1 && strcmp(bdroot,'ltpda_library')
0133                 HelpCallback();
0134                 return;
0135             end
0136             if value==1 && isempty(findobj('Name',tabbedFigName))
0137                 return;
0138             end
0139             otherwise
0140                 eval([property '=' value ';']);
0141             end
0142         end
0143
0144         parent = findobj('Name',tabbedFigName);
0145         if ~isempty(parent), clf(parent); set(parent,'Resize','on','ResizeFcn',{@GUIResize,guiSize});
0146         else parent =
figure('Position',guiSize,'Name',tabbedFigName,'Resize','on','ResizeFcn',{@GUIResize,guiSize},'NumberTitle','off','Toolbar','none','Menubar',
);
0147     end
0148
0149     set(parent,'Tag','LTPDAGUI','NextPlot','new');
0150
0151     if ~exist('margins','var'), margins = {1,'pixels'}; end
0152     if ~iscell(margins), margins = {margins,units}; end
0153     if ~exist('bordertype','var'), bordertype = 'none'; end
0154     if ~exist('titlebackgroundcolor','var'), titlebackgroundcolor = 0.95*panelbackgroundcolor; end
0155     if ~active, active = 1; end
0156
0157     clear htab
0158     htab = uipanel( ...
0159         'Parent',parent, ...
0160         'Units',units, ...
0161         'Position',position, ...
0162         'BackgroundColor',backgroundcolor, ...
0163         'BorderType',bordertype, ...
0164         'Visible','on', ...
0165         'Tag',tag);
0166
0167     set(htab,'Units','pixels');
0168     tabpos = get(htab,'Position');
0169
0170
0171     status = uicontainer( ...
0172         'Parent',htab, ...
0173         'BackgroundColor',backgroundcolor, ...
0174         'Units',margins{2}, ...
0175         'Position',[1,1,1,1], ...
0176         'Visible','off');
0177
0178     set(status,'Units','pixels');
0179     charsz = get(status,'Position');

```



```

0180
0181 margins = [charsz(1:2),charsz(1:2)].*margins{1};
0182 set(status,'Units','characters');
0183 set(status,'Position',[1,1,1,1]);
0184 set(status,'Units','pixels');
0185 charsz = get(status,'Position');
0186 titleheight = 1.6*charsz(4);
0187 titleheight = titleheight*scaleX;
0188
0189 margins(3:4) = -(margins(1:2)+margins(3:4));
0190 [loop,sign] = deal(1:ntab,[1,ntab,1,0]);
0191
0192 addtitle = 2*[sign(3:4),-1,-1];
0193 tttextpos = [0.05,0.08,0.9,0.76];
0194
0195 for kk=1:ntab
0196     title(kk) = uipanel( ...
0197         'Parent',htab, ...
0198         'Units','pixels', ...
0199         'BackgroundColor',titlebackgroundcolor, ...
0200         'BorderType','beveledout', ...
0201         'ButtonDownFcn',{@TopBottomTabCbk,kk,1}, ...
0202         'Visible','on');
0203
0204     ttext(kk) = uicontrol( ...
0205         'Parent',title(kk), ...
0206         'Units','normalized', ...
0207         'Position',tttextpos, ...
0208         'ForegroundColor',foregroundcolor, ...
0209         'BackgroundColor',titlebackgroundcolor, ...
0210         'HorizontalAlignment',horizontalalignment, ...
0211         'Style','text', ...
0212         'String',string{kk}, ...
0213         'FontSize',8*scaleX, ...
0214         'Enable','inactive', ...
0215         'HitTest','off', ...
0216         'Visible','on');
0217
0218 end
0219
0220 set(htab,'Units','pixels');
0221 dimension = get(htab,'Position');
0222 dimension(4) = dimension(4)-nrows*titleheight+5;
0223 panel =
uipanel('Parent',htab,'Units','pixels','Position',dimension,'BackgroundColor',panelbackgroundcolor,'BorderType','beveledout','Visible','on'
);
0224
0225 dimension = get(htab,'Position');
0226 logosize = [52,52];
0227 dimension = [dimension(3)-logosize(1)-3 , dimension(4)-logosize(2) , logosize];
0228 logo = axes('Parent',htab,'Units','pixels','Position',dimension,'Tag','Logo');
0229 image(imread('iconltp.jpg'),'Parent',logo,'AlphaData',imread('iconltpalpha.gif'));
0230 axis(logo,'off'); axis(logo,'square');
0231 set(logo,'HandleVisibility','callback');
0232
0233 dimension = get(htab,'Position');
0234 logosize = [40 40];
0235 dimension = [dimension(3)-logosize(1)-70 , dimension(4)-logosize(2)-9 , logosize];
0236 start2 = axes('Parent',htab,'Units','pixels','Position',dimension,'Tag','Start2');
0237 image(imread('startgreen.jpg'),'Parent',start2,'AlphaData',imread('startalpha.gif'),'ButtonDownFcn','ltpdagui(''redraw'',1.1);','Tag','StartBu
');
0238 axis(start2,'off');
0239 set(start2,'HandleVisibility','callback');
0240
0241 if active>ntab, active=ntab; end
0242

```

```

0243 inset(1) = uicontrol( ...
0244     'Parent', htab, ...
0245     'Units', 'pixels', ...
0246     'BackgroundColor', panelbackgroundcolor, ...
0247     'Style', 'text', ...
0248     'Visible', 'on');
0249 if ~isnan(titlehighlightcolor)
0250     inset(2) = uicontrol( ...
0251         'Parent', htab, ...
0252         'Units', 'pixels', ...
0253         'BackgroundColor', titlehighlightcolor, ...
0254         'Style', 'text', ...
0255         'Visible', 'on');
0256     inset(3) = uicontrol( ...
0257         'Parent', htab, ...
0258         'Units', 'pixels', ...
0259         'BackgroundColor', 0.85*titlehighlightcolor, ...
0260         'Style', 'text', ...
0261         'Visible', 'on');
0262 end
0263
0264 % =====
0265 % OTHER BUTTONS
0266 for xx=2:nrows
0267     localString = labels{xx};
0268     ntab = numel(localString);
0269     for tt=1:ntab
0270         titles(xx,tt) = uipanel( ...
0271             'Parent', htab, ...
0272             'Units', 'pixels', ...
0273             'BackgroundColor', titlebackgroundcolor, ...
0274             'BorderType', 'beveledout', ...
0275             'ButtonDownFcn', {@TopBottomTabCbK, tt, xx}, ...
0276             'Visible', 'on');
0277
0278         uicontrol( ...
0279             'Parent', titles(xx,tt), ...
0280             'Units', 'normalized', ...
0281             'Position', ttextpos, ...
0282             'ForegroundColor', foregroundcolor, ...
0283             'BackgroundColor', titlebackgroundcolor, ...
0284             'HorizontalAlignment', horizontalalignment, ...
0285             'Style', 'text', ...
0286             'String', localString{tt}, ...
0287             'Enable', 'inactive', ...
0288             'FontSize', 8*scaleX, ...
0289             'HitTest', 'off', ...
0290             'Visible', 'on');
0291     end
0292 end
0293
0294 % =====
0295
0296 TopBottomTabResizeCbK();
0297
0298 panelTitle = get(ttext(active), 'string');
0299 setappdata(0, 'ltpda_currPanel', panel);
0300
0301 % =====
0302 % =====
0303 % =====
0304 CheckTimerAlive()
0305
0306 switch panelTitle
0307     case 'Main LTPDA panel'
0308         if exist('runSim', 'var') && runSim==1
0309             runSim = 0; %#ok<NASGU>

```

```

0310         pan1main(panel,1)
0311     else pan1main(panel)
0312     end
0313     case 'Parameters'
0314         pan2param(panel)
0315     case 'Repository'
0316         pan3repo(panel)
0317     case 'Import from workspace'
0318         pan4import(panel)
0319     case 'Output'
0320         pan5output(panel)
0321     case 'Globals'
0322         pan6globals(panel)
0323     case 'Partial execution'
0324         pan7partial(panel)
0325     case 'Nested loops'
0326         panxyettocome(panel)
0327     % pan8nested(panel)
0328
0329 end
0330
0331 % =====
0332 % =====
0333 % =====
0334
0335
0336 %%
0337 %-----
0338 function TopBottomTabCbk(hobj,evdt,indx,numbrow) %#ok<INUSL>
0339 % Executed when the user clicks on a another tab or resize.
0340
0341 if numbrow>1 % the user clicked on another row
0342     for pp=ntab:-1:1, ntab = ntab - strcmp(string(pp),''); end
0343     if ntab==1, active=1; end
0344
0345     % Swop rows:
0346     temp = labels(1);
0347     labels(1) = labels(numbrow);
0348     labels(numbrow) = temp;
0349     clear temp
0350     % Rebuild the proper list:
0351     for ii=1:nrows
0352         temp(ii,1:numel(labels{ii}))=labels{ii};
0353     end
0354 else
0355     active = indx;
0356     for ii=1:nrows
0357         temp(ii,1:numel(labels{ii}))=labels{ii};
0358     end
0359 end
0360
0361 % =====
0362 ltpdagui('title',temp,'active',indx);
0363 % =====
0364
0365 end
0366 %-----
0367
0368 %-----
0369 function TopBottomTabResizeCbk(varargin)
0370 % Properly draws the panels.
0371
0372 units = get(htab,'Units');
0373 set(htab,'Units','pixels');
0374 tabpos = get(htab,'Position');
0375 tabpos(3) = tabpos(3)*0.85;
0376 set(htab,'Units',units);

```

```

0377 ntab = numel(string);
0378 for oo=ntab:-1:1, ntab = ntab - strcmp(string(oo), ''); end
0379
0380 titlepos = [0,0,0,titleheight];
0381 panelpos = [0,0,tabpos(3),tabpos(4)-titleheight]+margins;
0382 titlepos(2) = panelpos(2)+panelpos(4);
0383 panelpos(2) = panelpos(2)-titleheight*(nrows-1)+6;
0384
0385 n = sum(cellfun('length',string))+2*ntab;
0386 titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
0387 upperpos = titlepos(2);
0388 for k=loop
0389     titlepos(3) = panelpos(3)*(length(string{k})+2)/n;
0390     addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
0391     titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
0392     titlepos(2) = upperpos - titlepos(4)*(nrows-1)+3+1;
0393     set(title(k), 'Position', titlepos+addtopos);
0394     titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
0395 end
0396
0397 % =====
0398
0399 if nrows>1
0400     for xxx=2:nrows
0401         localString = labels{xxx};
0402         ntab = numel(localString);
0403         for i=ntab:-1:1, ntab = ntab - strcmp(localString(i), ''); end
0404         n = sum(cellfun('length',localString))+2*ntab;
0405         titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
0406         titlepos(2) = titlepos(2) + titlepos(4)-2;
0407         for k=1:ntab
0408             titlepos(3) = panelpos(3)*(length(localString{k})+2)/n;
0409             addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
0410             titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
0411             if ntab==1, titlepos(3)=titlepos(3)-2; end
0412             set(titles(xxx,k), 'Position', titlepos+addtopos);
0413             titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
0414         end
0415         if ntab<numel(localString)
0416             for qq=ntab+1:numel(localString)
0417                 nullpos=[0,0,1,1];
0418                 set(titles(xxx,qq), 'Position', nullpos);
0419             end
0420         end
0421     end
0422 end
0423
0424 % =====
0425
0426 titlepos = get(title(active), 'Position');
0427 titlepos = titlepos-addtitle.*[active==sign(1),1,active==sign(1),1];
0428 titlepos(3) = titlepos(3)-addtitle(3)*(active==sign(2));
0429 ntab = numel(string);
0430 for i=ntab:-1:1, ntab = ntab - strcmp(string(i), ''); end
0431 if ntab==1, titlepos(3) = titlepos(3)-4; end
0432
0433 set(title(active), 'Position', titlepos, 'BackgroundColor', panelbackgroundcolor);
0434 set(ttext(active), 'FontWeight', 'bold', 'BackgroundColor', panelbackgroundcolor);
0435
0436 insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
0437 set(inset(1), 'Position', insetpos);
0438 try
0439     insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
0440     set(inset(2), 'Position', insetpos);
0441     set(inset(3), 'Position', insetpos+[2,1-2*sign(4),-4,0]);
0442 catch
0443 end

```

```

0444     end
0445     %-----
0446
0447     %-----
0448     function GUIResize(varargin)
0449     % Called whenever the user resize the GUI window.
0450
0451         guiSize = varargin{3};
0452         newSize = get(findobj('Tag','LTPDAGUI'),'Position');
0453
0454         % To avoid a width lower than the fixed minimum:
0455         if newSize(3)<guiSize(3), newSize(3)=guiSize(3); end
0456         % To avoid an height lower than the fixed minimum:
0457         if newSize(4)<guiSize(4), newSize(4)=guiSize(4); end
0458         % To avoid a window moved outside the screen border:
0459         screenSize = get(0,'screensize');
0460         if newSize(2)+newSize(4)+30>screenSize(4), newSize(2)=screenSize(4)-newSize(4)-40; end
0461         if newSize(1)+newSize(3)>screenSize(3), newSize(1)=screenSize(3)-newSize(3)-5; end
0462
0463         set(findobj('Tag','LTPDAGUI'),'Position',newSize);
0464
0465         % Redraw the GUI on the current panel:
0466         pause(0.1)
0467         if ~getappdata(0,'startup'), TopBottomTabCbK([],[],active,1)
0468         else setappdata(0,'startup',0);
0469         end
0470
0471     end
0472     %-----
0473
0474 %%
0475 %-----
0476     function CheckTimerAlive(varargin)
0477     % This is the function to execute a continuous timer check
0478
0479     % Keep alive the first timer:
0480     alltimers = timerfindall;
0481     if strcmp(get(alltimers(1),'Running'),'off')
0482         start(alltimers(1))
0483     end
0484 end
0485 %-----
0486
0487 %-----
0488     function HelpCallback(varargin)
0489     % Whenever the user doubleclick a block in the library window (ie, the
0490     % OpeFcn callback is executed from the library).
0491
0492     try
0493         childpath = find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');
0494         functionname = get_param(childpath,'Tag');
0495
0496         description = get_param(gcbh,'Description');
0497         funcname = findstr('functionName=',description);
0498         if funcname
0499             i = funcname+14;
0500             while ~strcmp(description(i),';'), i=i+1; end
0501             functionname = description(funcname+14:i-2);
0502         end
0503
0504         [func,classes] = strtok(get(gcbh,'Tag'));
0505         if strcmp(func,'method')
0506             helptext = help([strtrim(classes),'/',functionname]);
0507         else
0508             helptext = help(functionname);
0509         end
0510

```

```
0511         helpFigure = figure('MenuBar','none','ToolBar','none','Name',strcat(functionname,'
help'),'NumberTitle','off','Units','pixels','Position',[200,200,500,300],'Resize','on','ResizeFcn','refresh(gcf)','Tag','helpwindow','Color'
, get(0, 'defaultuicontrolbackgroundcolor'));
0512         % Help text:
0513         uicontrol('Parent',helpFigure,'Units','normalized','HorizontalAlignment','left','Position',[0 0 1
1],'FontSize',8,'String',helptext,'Visible','on','max',100,'Style','edit');
0514     catch
0515     end
0516 end
0517 %-----
0518
0519 end
```

[Home](#) > [m](#) > [gui](#) > [gltpda](#) > ltpdalib.m

ltpdalib

PURPOSE [^]

LTPDALIB Open LTPDA Package Library.

SYNOPSIS [^]

function ltpdalib()

DESCRIPTION [^]

LTPDALIB Open LTPDA Package Library.

LTPDALIB opens the latest version of LTPDA Library Blockset.

Other information available for the LTPDA Blockset:
 help ltpda - to view the Contents file

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SOURCE CODE [^]

```

0001 function ltpdalib()
0002 %LTPDALIB Open LTPDA Package Library.
0003 %
0004 % LTPDALIB opens the latest version of LTPDA Library Blockset.
0005 %
0006 %
0007 % Other information available for the LTPDA Blockset:
0008 % help ltpda - to view the Contents file
0009 %
0010 %
0011 % N Tateo
0012 % $Id: ltpdalib.m,v 1.1 2008/03/01 13:43:20 nicola Exp $ $Date: 2008/03/01 13:43:20 $
0013 %
0014 if ~isempty(nargchk(0,0,nargin))
0015     error('ltpda:ltpdalib:invalidnumberinputs','Too many inputs');
0016 end
0017 %
0018 % Attempt to open library:
0019 model = 'ltpda_library';
0020 %
0021 try
0022     open_system(model);
0023 catch
0024     error('ltpda:ltpdalib:invalidmodelname',['Could not find LTPDA Library (' model
0025     '.mdl).']);
0026 end
0027 % end of ltpdalib.m

```


ltpdasim

PURPOSE ^

This is the automatic function wrapper

SYNOPSIS ^

function ltpdasim(block)

DESCRIPTION ^

```

    This is the automatic function wrapper
=====
===== level-2 M file S-function =====
=====
This wrapper is the automatic function, called by every function block
in Simulink, able to execute m-file functions retrieving from Simulink:
(1) the pointer to the object(s) to be analyzed, coming in as input of
    the corresponding block (ie, the DATA),
(2) the name of the function to be applied on those data, from the tag
    of the currently executed block (ie, the true FUNCTION),
(3) the parameters for that particular block, retrieved from the global
    shared workspace by the handle of the block (ie, the PARAMETERS).

The output is then generated as:
    OUTPUT = FUNCTION(DATA,PARAMETERS)

This output in the end is saved into the global array containing all
the AOs (ie, all the DATA go together with other data): thus this output
will be freely accessible by all the other functions.

The only real output to Simulink will be just the ordinal number of the
so-generated AO into the global array of AOs.

    $Id: ltpdasim.m,v 1.24 2008/08/25 13:20:17 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [progressBar](#) Progress bar for the LTPDA GUI.

This function is called by:

SUBFUNCTIONS ^

-  [function setup\(block\)](#)
-  [function SetInpPortFrameData\(block, idx, fd\)](#)
-  [function SetInpPortDims\(block, idx, di\)](#)
-  [function Outputs\(block\)](#)

SOURCE CODE ^

```
0001 function ltpdasim(block)
0002
0003 %           This is the automatic function wrapper
0004 % =====
0005 % ===== level-2 M file S-function =====
0006 % =====
0007 % This wrapper is the automatic function, called by every function block
0008 % in Simulink, able to execute m-file functions retrieving from Simulink:
0009 % (1) the pointer to the object(s) to be analyzed, coming in as input of
```

```

0010 % the corresponding block (ie, the DATA),
0011 % (2) the name of the function to be applied on those data, from the tag
0012 % of the currently executed block (ie, the true FUNCTION),
0013 % (3) the parameters for that particular block, retrieved from the global
0014 % shared workspace by the handle of the block (ie, the PARAMETERS).
0015 %
0016 % The output is then generated as:
0017 %     OUTPUT = FUNCTION(DATA,PARAMETERS)
0018 %
0019 % This output in the end is saved into the global array containing all
0020 % the AOs (ie, all the DATA go together with other data): thus this output
0021 % will be freely accessible by all the other functions.
0022 %
0023 % The only real output to Simulink will be just the ordinal number of the
0024 % so-generated AO into the global array of AOs.
0025 %
0026 % $Id: ltpdasim.m,v 1.24 2008/08/25 13:20:17 nicola Exp $
0027
0028
0029 setup(block);
0030
0031
0032 %% Setup
0033
0034 function setup(block)
0035
0036 % Register dialog parameter: none, because they're retrieved directly
0037 % from the memory. This will prevent the user to modify the parameters
0038 % outside the proper parameters panel:
0039 block.NumDialogPrms = 0;
0040
0041 % Register number of input and output ports
0042 block.NumInputPorts = 1;
0043 block.NumOutputPorts = 1;
0044
0045 % Setup functional port properties to dynamically inherited.
0046 block.SetPreCompInpPortInfoToDynamic;
0047
0048 block.InputPort(1).DirectFeedthrough = true;
0049 block.InputPort(1).DatatypeID = 0;
0050 block.InputPort(1).Complexity = 0;
0051 block.OutputPort(1).DatatypeID = 0;
0052 block.OutputPort(1).Complexity = 0;
0053 block.SampleTimes = [0 0];
0054 block.SetAccelRunOnTLC(false);
0055
0056 block.OutputPort(1).Dimensions = 1;
0057
0058 % Register methods
0059 block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
0060 block.RegBlockMethod('SetInputPortDimensions',@SetInpPortDims);
0061 block.RegBlockMethod('Outputs',@Outputs);
0062
0063 function SetInpPortFrameData(block, idx, fd) %#ok<INUSL>
0064 block.InputPort(1).SamplingMode = fd;
0065 block.OutputPort(1).SamplingMode = fd;
0066
0067 function SetInpPortDims(block, idx, di)
0068 block.InputPort(idx).Dimensions = di;
0069
0070
0071 %% Outputs
0072 function Outputs(block)
0073 global LTPDAinvar gl execModels %#ok<NUSED>
0074
0075 % disp(['The name of the currently executed block is
0076 %',get_param(get_param(gcbh,'Parent'),'Name')])
0076 % disp(['The current size of LTPDAinvar is ',num2str(size(LTPDAinvar,1))])
0077
0078 currparent = get_param(gcbh,'Parent');
0079 subsyshandle = get_param(currparent,'handle');
0080
0081 % Check if it's a partial execution:
0082 if ismember('executionList',evalin('base','who'))
0083     executionList = evalin('base','executionList');
0084     blockIndex = -1;
0085     for i=1:size(executionList,1), if executionList(i)==gcbh, blockIndex = i; break; end;
0086 end
0086 if blockIndex == -1 % this block is not included in the list of those to be executed.
0087     disp(['. Partial execution: block ',get_param(subsyshandle,'Name'),' will be
0088     skipped.'])
0088     block.OutputPort(1).Data = -1;
0089     return
0090 else
0091     % Check if the block must receive data from Simulink or retrieve them from a
0092     previous calculation:
0092     if block.InputPort(1).Data(1)==-1

```

```

0093     annotation = find_system(bdroot, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda
model');
0094     execHistory = get_param(annotation, 'UserData');
0095     blockIndex = -1;
0096     for i=1:size(execHistory,1), if execHistory{i,2}==subsyshandle, blockIndex = i;
end; end
0097     if blockIndex == -1 % something's wrong: the selected block is not included in
the previous execution history
0098         warning(['*** Partial execution: impossible to retrieve inputs. The block
', get_param(subsyshandle, 'Name'), ' seems to have never been executed.'])
0099         block.OutputPort(1).Data = -1;
0100         return
0101     end
0102     j=3;
0103     while ~isempty(execHistory{blockIndex, j})
0104         block.InputPort(1).Data(j-2) = execHistory{blockIndex, j};
0105         j = j+1;
0106     end
0107     disp([' . Partial execution: block ', get_param(subsyshandle, 'Name'), ' will now
be executed with data retrieved from previous calculation.'])
0108     else
0109         disp([' . Partial execution: block ', get_param(subsyshandle, 'Name'), ' will now
be executed.'])
0110     end
0111 end
0112 end
0113
0114 % Check if the user wants to stop the execution:
0115 lastChar = get(findobj('Name', 'LTPDA Progress Bar'), 'CurrentCharacter');
0116 if ~isempty(lastChar) && strcmp(lastChar, 'x')
0117     set_param(bdroot, 'SimulationCommand', 'stop')
0118     return
0119 end
0120
0121 % Update the progress bar window:
0122 progressBar = findobj('Tag', 'progressaxes');
0123 if ~isempty(progressBar)
0124     blocksTotal = get(findobj('Tag', 'blockstotal'), 'UserData');
0125     blocksDone = get(findobj('Tag', 'done'), 'UserData');
0126     blocksToGo = get(findobj('Tag', 'togo'), 'UserData');
0127     set(findobj('Tag', 'done'), 'UserData', blocksDone+1);
0128     set(findobj('Tag', 'togo'), 'UserData', blocksToGo-1);
0129     set(findobj('Tag', 'done'), 'String', ['Done: ', num2str(blocksDone+1)]);
0130     set(findobj('Tag', 'togo'), 'String', ['To go: ', num2str(blocksToGo-1)]);
0131     set(findobj('Tag', 'currentexec'), 'String', ['Currently executed block:
', get_param(get_param(gcbbh, 'Parent'), 'Name')]);
0132     progressSize = get(progressBar, 'UserData');
0133     progressPosition = get(progressBar, 'Position');
0134     progressPosition(3) = 1+(progressSize(3)/blocksTotal)*(blocksDone+1);
0135     set(progressBar, 'Position', progressPosition)
0136     drawnow
0137 end
0138
0139 % Retrieve the function name from the block tag:
0140 blocktag = lower(get_param(gcbbh, 'tag'));
0141 if strcmp(blocktag, 'generic')
0142     paramcommand = get_param(get_param(get_param(gcbbh, 'Parent'),
'handle'), 'Description');
0143     eval(paramcommand)
0144     blocktag = functionName;
0145 end
0146
0147 % =====
0148 %% For arithmetic blocks
0149 % =====
0150
0151 if strcmp(blocktag, 'arithmetic')
0152     eq = get_param(currparent, 'MaskDescription');
0153     if strcmp(eq, 'Arithmetic block: the user can type in the desired equation.')
0154         disp(sprintf('Error: in the ''s'' arithmetic block no equation has been typed
in.', get_param(currparent, 'Name')))
0155         close(findobj('Name', 'LTPDA Progress Bar'))
0156         error(['Error: in the ', get_param(currparent, 'Name'), ' arithmetic block no
equation has been typed in.'])
0157     end
0158     equation = eq;
0159     equation(strfind(equation, char(10))) = [];
0160
0161     paramcommand = get_param(subsyshandle, 'Description');
0162     if isempty(paramcommand), paramcommand = 'params=plist('alpha', '--
>', 'beta', '-->'); end
0163     eval(paramcommand)
0164
0165     for i=1:nparams(params) %#ok<NODEF>
0166         inputBlock =
find_system(currparent, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'Name'
, lower(params.params(i).key));

```

```

0167         if numel(inputBlock)>1, inputBlock(1)=[]; end % this can happen only when the
currparent has the same name: then, ignore the 1st, ie. the currparent itself
0168         inputNumb = get_param(inputBlock{1}, 'Tag');
0169         equation =
strrep(equation, lower(params.params(i).key), ['LTPDAinvar{block.InputPort(1).Data(' ,inputNumb, '),1,1
]);
0170     end
0171
0172     try
0173         outdata = eval(equation);
0174
0175         % Setting the output signal name:
0176         ifsetName = get_param(currparent, 'UserData');
0177         if ~isempty(ifsetName) && isa(ifsetName, 'char')
0178             % ifsetName = ifsetName;
0179         else
0180             ifsetName = get_param(currparent, 'Name');
0181         end
0182         if numel(outdata)==1
0183             outdata = setName(outdata, ifsetName);
0184         else
0185             for i=1:size(outdata,1)
0186                 for j=1:size(outdata,2)
0187                     outdata(i,j) = setName(outdata(i,j), [ifsetName, '_', num2str(i), '-'
num2str(j)]);
0188                 end
0189             end
0190         end
0191
0192         x = size(LTPDAinvar,1);
0193         LTPDAinvar = [LTPDAinvar;{outdata,0}];
0194         block.OutputPort(1).Data = x+1;
0195
0196         probe = get_param(currparent, 'MaskHelp');
0197         if ~isempty(probe) && isa(probe, 'char') && strcmp(probe, 'probe')
0198             LTPDAinvar{size(LTPDAinvar,1),2} = 2;
0199         end
0200     return
0201
0202     catch err
0203         disp(sprintf('Something''s wrong in the calculation of the results of the ''%s''
arithmetic block.', get_param(currparent, 'Name')))
0204         disp('The inputs were:')
0205         for i=1:nparams(params)
0206             disp(['Input ', num2str(i), ':'])
0207             LTPDAinvar{block.InputPort(1).Data(i),1} %#ok<NOPRT>
0208         end
0209         disp('The equation was:')
0210         disp(eq)
0211         close(findobj('Name', 'LTPDA Progress Bar'))
0212         rethrow(err)
0213
0214     end
0215 end % for the arithmetic block
0216
0217 % =====
0218 %% ===== RETRIEVE THE PARAMETERS LIST =====
0219 % =====
0220
0221 % Set this block status to 'executed':
0222 set(subsyshandle, 'MaskVariables', '1');
0223 set(gcbh, 'MaskVariables', '1');
0224
0225 % The command line to produce the proper parameters list for the block
0226 % currently executed is retrieved from the description of this parent subsystem
0227 paramcommand = get_param(subsyshandle, 'Description');
0228
0229 if ~isempty(paramcommand)
0230     eval(paramcommand);
0231
0232     % Verify if a global variable name has been used:
0233     for i=1:nparams(params) %#ok<NODEF>
0234         if ischar(params.params(i).val)
0235             try %#ok<ALIGN>
0236                 % params.params(i).val = evalin('base', params.params(i).val);
0237                 params.params(i).val = eval(params.params(i).val);
0238             catch end
0239         end
0240     end
0241
0242     if ~exist('paramEnabled', 'var'), paramEnabled = ones(1, nparams(params)); end
0243     currparam = plist();
0244     for jj=1:nparams(params)
0245         if paramEnabled(jj)==1
0246             try %#ok<ALIGN>

```

```

0249         if strcmpi(params.params(jj).key(1:7), 'addPar_'),
params.params(jj).key(1:7)=[]; end
0250     catch end
0251     try % for cell array params:
0252     %         if isa(params.params(jj).val,'char') &&
strcmp(params.params(jj).val(1),'{'), params.params(jj).val = eval(params.params(jj).val) ; end
0253     %         catch
0254     %         end
0255     currparam = append(currparam,params.params(jj));
0256     end
0257     end
0258     else
0259     currparam = plist();
0260     end
0261
0262     % Retrieve parameters from Simulink:
0263     paramFromSimulink = 0;
0264     for i=1:nparams(currparam)
0265         paramKey = currparam.params(i).key;
0266         paramVal = currparam.params(i).val;
0267         if (isa(paramVal,'char') && strcmp(paramVal,'-->')) || (isa(paramVal,'cell') &&
numel(paramVal)==1 && isa(paramVal{1},'char') && strcmp(paramVal{1},'-->'))
0268             paramFromSimulink = paramFromSimulink + 1;
0269             mux =
find_system(currparent, 'SearchDepth',1, 'LookUnderMasks', 'all', 'BlockType', 'Mux');
0270             muxWidths = get_param(mux{1}, 'CompiledPortWidths');
0271             muxWidths = muxWidths.Inputport;
0272             inputBlock =
find_system(currparent, 'SearchDepth',1, 'LookUnderMasks', 'all', 'Name', lower(paramKey));
0273             if numel(inputBlock)>1, inputBlock(1)=[]; end % this can happen only when
the currparent has the same name: then, ignore the 1st, ie. the currparent itself
0274             inputNumb = str2double(get_param(inputBlock{1}, 'Tag'));
0275             x = 0;
0276             for j=1:inputNumb-1
0277                 x=x+muxWidths(j);
0278             end
0279             paramVal = [LTPDAinvar{block.InputPort(1).Data(x+1)}];
0280             if muxWidths(inputNumb)>1
0281                 for j=x+2:x+muxWidths(inputNumb)
0282                     paramVal = [paramVal, LTPDAinvar{block.InputPort(1).Data(j)}];
0283                 end
0284             end
0285             currparam = pset(currparam,paramKey,paramVal);
0286         end
0287     end
0288
0289     % =====
0290     %% ===== CALCULATE THE RESULTS =====
0291     % =====
0292
0293     y = length(block.InputPort(1).Data) - paramFromSimulink;
0294
0295     % Check to verify the type of function:
0296     specFunc = {'display'};
0297     [meth,classes] = strtok(get(get_param(currparent, 'Handle'), 'Tag'));
0298     if strcmp(meth, 'method')
0299         clas = strtrim(classes);
0300         infoObj = eval([clas, '.getInfo(''', blocktag, ''')']);
0301         category = infoObj.mcategory;
0302     end
0303
0304     if ismember(blocktag, specFunc)
0305         % Input, but no parameters and no output
0306         type = 4;
0307         % =====
0308     elseif (exist('category', 'var') && strcmp(category, 'Output'))
0309         % Input and parameters, but no output
0310         type = 3;
0311         % =====
0312     elseif
~isempty(utils.prog.find_in_models(currparent, 'SearchDepth',1, 'LookUnderMasks', 'all', 'BlockType',
)) || block.InputPort(1).Data(1)==0
0313         % No input, but parameters and output
0314         type = 2;
0315         % =====
0316     else
0317         % Input, parameters and output
0318         type = 1;
0319         % =====
0320     end
0321
0322     switch type
0323     % =====
0324     case 4 % Input, but no parameters and no output
0325         command = 'feval(blocktag, '
0326         for i=1:y

```



```

0328         dVal = block.InputPort(1).Data(i);
0329         if dVal>0, command = [command sprintf('LTPDAinvar{%d}', dVal)]; end
0330     end
0331     command = [command(1:end-1) ' ');];
0332     try
0333         eval(command)
0334     catch err
0335         disp('4')
0336         disp(sprintf('Something''s wrong in the calculation of the results of the
0337 %s function.', blocktag))
0337         disp(sprintf('The name of the currently executed block is
0338 %s.', get_param(currparent, 'Name')))
0338         disp(sprintf('The (first) input data ordinal number, in the list of
0339 objects, was %d.', block.InputPort(1).Data(1)))
0339         close(findobj('Name', 'LTPDA Progress Bar'))
0340         rethrow(err)
0341     end
0342     block.OutputPort(1).Data = 1; % this will be ignored: no output expected.
0343     return;
0344
0345 % =====
0346 case 3 % Input and parameters, but no output
0347     try
0348         command = 'feval(blocktag, ';
0349         for i=1:y
0350             dVal = block.InputPort(1).Data(i);
0351             if dVal>0, command = [command sprintf('LTPDAinvar{%d}', dVal)]; end
0352         end
0353         command = [command 'currparam);'];
0354         if strcmpi(blocktag, 'plot'), figure; end;
0355         eval(command);
0356     catch err
0357         disp('3')
0358         disp(sprintf('Something''s wrong in the calculation of the results of the
0359 %s function.', blocktag))
0359         disp(sprintf('The name of the currently executed block is
0360 %s.', get_param(currparent, 'Name')))
0360         disp(sprintf('The (first) input data ordinal number, in the list of
0361 objects, was %d.', block.InputPort(1).Data(1)))
0361         disp('The plist passed was:')
0362         currparam %#ok<NOPRT>
0362         close(findobj('Name', 'LTPDA Progress Bar'))
0363         rethrow(err)
0364     end
0365     block.OutputPort(1).Data = 1; % this will be ignored: no output expected.
0366     return;
0367
0368 % =====
0369 case 2 % No input, but parameters and output
0370     try
0371         outdata = feval(blocktag, currparam);
0372     catch err
0373         disp('2')
0374         disp(sprintf('Something''s wrong in the calculation of the results of the
0375 %s function.', blocktag))
0375         disp(sprintf('The name of the currently executed block is
0376 %s.', get_param(currparent, 'Name')))
0376         disp('The plist passed was:')
0377         currparam %#ok<NOPRT>
0377         close(findobj('Name', 'LTPDA Progress Bar'))
0378         rethrow(err)
0379     end
0380
0381 % =====
0382 case 1 % Input, parameters and output
0383     try
0384         command = 'outdata = feval(blocktag, ';
0385         for i=1:y
0386             dVal = block.InputPort(1).Data(i);
0387             if dVal>0, command = [command sprintf('LTPDAinvar{%d}', dVal)]; end
0388         end
0389         command = [command 'currparam);'];
0390         eval(command);
0391     catch err
0392         disp('1')
0393         disp(sprintf('Something''s wrong in the calculation of the results of the
0394 %s function.', blocktag))
0394         disp(sprintf('The name of the currently executed block is
0395 %s.', get_param(currparent, 'Name')))
0395         disp(sprintf('The (first) input data ordinal number, in the list of
0396 objects, was %d.', block.InputPort(1).Data(1)))
0396         disp('The plist passed was:')
0397         currparam %#ok<NOPRT>
0397         close(findobj('Name', 'LTPDA Progress Bar'))
0398         rethrow(err)
0399     end
0400
0401
0402
0403

```

```

0404     end
0405
0406 % =====
0407 %% ===== OUTPUTS =====
0408 % =====
0409
0410 signalName = get_param(currparent, 'Name');
0411 ifsetName = get_param(currparent, 'UserData');
0412 if ~isempty(ifsetName) && isa(ifsetName, 'double') && ifsetName==1
0413     if numel(outdata)==1
0414         outdata = setName(outdata, signalName);
0415     else
0416         for i=1:size(outdata,1)
0417             for j=1:size(outdata,2)
0418                 outdata(i,j) = set(outdata(i,j), 'name', [signalName, '_', num2str(i), '-',
0419                     num2str(j)]);
0419             end
0420         end
0421     end
0422 end
0423
0424 try
0425 % To prevent the result calculated here from being canceled at the
0426 % end of the calculation:
0427 probe = get_param(currparent, 'MaskHelp');
0428 if ~isempty(probe) && isa(probe, 'char') && strcmp(probe, 'probe'), keepThis = 1;
0429 else keepThis = 0;
0430 end
0431
0432 % Appending the results:
0433 ltpda_annotation = find_system(bdroot, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda
model');
0434 execHistory = get_param(ltpda_annotation, 'UserData');
0435 if ~isempty(execHistory)
0436     member = 0;
0437     for i=1:size(execHistory,1)
0438         if execHistory{i,2}==subsyshandle, member = 1; index = i; break; end
0439     end
0440 end
0441 if ~isempty(execHistory) && member,
0442     x = execHistory{index,1};
0443     if x==-1, x = size(LTPDAinvar,1)+1; end
0444     LTPDAinvar(x,:) = {outdata, keepThis};
0445     block.OutputPort(1).Data = x;
0446     execHistory{index,1} = x;
0447     execHistory{index,2} = subsyshandle;
0448     for i=1:y, execHistory{index,2+i} = block.InputPort(1).Data(i); end
0449 else
0450     if isempty(execHistory), execHistory = {}; end
0451     x = size(LTPDAinvar,1);
0452     LTPDAinvar = [LTPDAinvar; {outdata, keepThis}];
0453     block.OutputPort(1).Data = x+1;
0454     execHistory{end+1,1} = x+1;
0455     execHistory{end,2} = subsyshandle;
0456     for i=1:y, execHistory{end,2+i} = block.InputPort(1).Data(i); end
0457 end
0458
0459 % Update the execution history:
0460 % Are stored the the output obj pointer, the handle of the current subsystem and the
input obj pointers:
0461 set_param(ltpda_annotation, 'UserData', execHistory);
0462
0463
0464 catch
0465     disp('-----')
0466     disp('-----')
0467     disp('-----')
0468     disp(sprintf('Something's wrong in appending to the global variable LTPDAinvar the
results of the %s function.', blocktag))
0469     disp(sprintf('The name of the currently executed block is
%s.', get_param(currparent, 'Name')))
0470     disp('The data to be saved, calculated into this block, are:')
0471     outdata %#ok<NOPRT>
0472     disp('-----')
0473     disp('-----')
0474     disp('-----')
0475 end
0476
0477 %=====
0478 %% Cycle to remove objects from memory as soon as they're no longer necessary:
0479 %=====

```

```

0480
0481 if ~getappdata(0, 'maintainresults');
0482     portConnectivity = get(subsyshandle, 'PortConnectivity');
0483     for i = 1:numel(portConnectivity)
0484         notUsedAnymore = 1;
0485         if isempty(portConnectivity(i).DstBlock)
0486             parentBlock = portConnectivity(i).SrcBlock;
0487             childrenBlocks = utils.prog.findchildren(parentBlock);
0488             for j = 1:numel(childrenBlocks)
0489                 if isempty(get_param(childrenBlocks(j), 'MaskVariables')), notUsedAnymore = 0;
0490             end
0491         end
0492         if notUsedAnymore && LTPDAinvar{block.InputPort(1).Data(i),2}==0
0493             LTPDAinvar(block.InputPort(1).Data(i),:) = [];
0494         end
0495         % Update the execution history:
0496         ltpda_annotation =
0497         find_system(bdroot, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda model');
0498         execHistory = get_param(ltpda_annotation, 'UserData');
0499         for xx=1:size(execHistory,1)
0500             for yy=3:size(execHistory,2),
0501                 if execHistory{xx,yy} == block.InputPort(1).Data(i), execHistory{xx,yy}
0502                     = -1; end
0503                 if execHistory{xx,yy} > block.InputPort(1).Data(i), execHistory{xx,yy}
0504                     = execHistory{xx,yy}-1; end
0505             end
0506             if execHistory{xx,1} == block.InputPort(1).Data(i), execHistory{xx,1} = -
0507                 1; end
0508             if execHistory{xx,1} > block.InputPort(1).Data(i), execHistory{xx,1} =
0509                 execHistory{xx,1}-1; end
0510             end
0511             set_param(ltpda_annotation, 'UserData', execHistory);
0512             disp('*** LTPDA object cleared from memory, being now unused')
0513         end
0514     end
0515 end
0516 %=====
0517 % The output to Simulink is just the ordinal number (ONE AND ONLY ONE
0518 % NUMBER) of the object appended to the global array LTPDAinvar.
0519 % endfunction

```


[Home](#) > [m](#) > [gui](#) > [gltpda](#) > pan1main.m

pan1main

PURPOSE ^

This is the function for the main panel of the GUI LTPDA, the package of

SYNOPSIS ^

function pan1main(varargin)

DESCRIPTION ^

This is the function for the main panel of the GUI LTPDA, the package of data analysis functions developed for the LISA Pathfinder Mission / LISA Technology Package.

\$Id: pan1main.m,v 1.38 2008/08/30 12:55:42 nicola Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [loginLTPDA](#) LOGINDLG Dialog for visually secure login.
-  [ltpdagui](#) =====
-  [progressBar](#) Progress bar for the LTPDA GUI.
-  [uipickfiles](#) uipickfiles: GUI program to select file(s) and/or directories.

This function is called by:

-  [ltpdagui](#) =====

SUBFUNCTIONS ^

-  [function ContinuousCheck\(varargin\)](#)
-  [function LogoutCallback\(varargin\)](#)
-  [function AskConfirm\(varargin\)](#)
-  [function hLoadDataButtonCallback\(varargin\)](#)
-  [function hRemoveButtonCallback\(varargin\)](#)
-  [function hExportButtonCallback\(varargin\)](#)
-  [function hExploreButtonCallback\(varargin\)](#)
-  [function SaveWorkspaceCallback\(varargin\)](#)
-  [function LoadWorkspaceCallback\(varargin\)](#)
-  [function UpdateNestedLoops\(hObject,varargin\)](#)
-  [function UpdateListbox\(varargin\)](#)
-  [function hListObjsCallback\(hObject, varargin\)](#)
-  [function hNewButtonCallback\(varargin\)](#)
-  [function hSaveCallback\(varargin\)](#)
-  [function hOpenCallback\(varargin\)](#)
-  [function StartExec\(varargin\)](#)

SOURCE CODE ^

```
0001 function pan1main(varargin)
0002 % This is the function for the main panel of the GUI LTPDA, the package of
0003 % data analysis functions developed for the LISA Pathfinder Mission / LISA
0004 % Technology Package.
0005
0006 % $Id: pan1main.m,v 1.38 2008/08/30 12:55:42 nicola Exp $
0007
0008 global LTPDAinvar LTPDAoutvar currentssystem gl
0009
0010 % Initialize the list of AOs:
0011 InputObjList=''; % OutputObjList='';
0012 UpdateListbox();
0013
0014 scaleX = getappdata(0, 'ltpda_gui_scalefactor');
0015 guiFontSize = round(8*scaleX);
0016
0017 alltimers = timerfindall;
0018 stop(alltimers(2));
0019 set(alltimers(2), 'TimerFcn', @ContinuousCheck, 'Period', .5);
0020 start(alltimers(2));
0021 set(gcf, 'NextPlot', 'new');
0022 set(gcf, 'CloseRequestFcn', {@AskConfirm, 'close'})
0023
0024 %%
0025
0026 currPanel = varargin{1};
0027 if nargin>1, runSim = varargin{2}; else runSim = 0; end
0028 panelDimens = get(currPanel, 'Position');
0029 backColor = get(currPanel, 'BackgroundColor');
0030
0031 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'center', 'Units', 'normalized', 'Position', [0.0375 0.895 0.075 0.06], 'String', 'Username:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0032 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'center', 'Position', [0.1187 0.9213 0.1
0.050], 'String', getappdata(0, 'ltpda_repo_user'), 'FontSize', guiFontSize, 'FontAngle', 'italic', 'Visible', 'on', 'Enable', 'off', 'Callback', @UserIDCallback, 'Style', 'edit');
0033 uicontrol('Parent', currPanel, 'Units', 'normalized', 'HorizontalAlignment', 'center', 'Position', [0.2375 0.9274 0.15 0.040], 'String', 'Login / Logout', 'FontSize', guiFontSize, 'Visible', 'on', 'Enable', 'on', 'Callback', @LogoutCallback, 'Style', 'pushbutton');
0034
0035 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'center', 'Units', 'normalized', 'Position', [0.5 0.915 0.125 0.040], 'String', 'Current system:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0036 currentssystem = 'brroot';
0037 if isempty(currentssystem) || isempty(find_system(currentssystem, 'FindAll', 'on', 'Type', 'ltpda_model')), currentssystem = 'None'; end
0038 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'center', 'Units', 'normalized', 'Position', [0.6375 0.915 0.1875 0.040], 'String', currentssystem, 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'UserData', 'currentssystem', 'Style', 'text');
0039
0040
0041
0042
0043
0044
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054
0055
0056
0057
0058
0059
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
0100
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110
0111
0112
0113
0114
0115
0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172
0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187
0188
0189
0190
0191
0192
0193
0194
0195
0196
0197
0198
0199
0200
0201
0202
0203
0204
0205
0206
0207
0208
0209
0210
0211
0212
0213
0214
0215
0216
0217
0218
0219
0220
0221
0222
0223
0224
0225
0226
0227
0228
0229
0230
0231
0232
0233
0234
0235
0236
0237
0238
0239
0240
0241
0242
0243
0244
0245
0246
0247
0248
0249
0250
0251
0252
0253
0254
0255
0256
0257
0258
0259
0260
0261
0262
0263
0264
0265
0266
0267
0268
0269
0270
0271
0272
0273
0274
0275
0276
0277
0278
0279
0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343
0344
0345
0346
0347
0348
0349
0350
0351
0352
0353
0354
0355
0356
0357
0358
0359
0360
0361
0362
0363
0364
0365
0366
0367
0368
0369
0370
0371
0372
0373
0374
0375
0376
0377
0378
0379
0380
0381
0382
0383
0384
0385
0386
0387
0388
0389
0390
0391
0392
0393
0394
0395
0396
0397
0398
0399
0400
0401
0402
0403
0404
0405
0406
0407
0408
0409
0410
0411
0412
0413
0414
0415
0416
0417
0418
0419
0420
0421
0422
0423
0424
0425
0426
0427
0428
0429
0430
0431
0432
0433
0434
0435
0436
0437
0438
0439
0440
0441
0442
0443
0444
0445
0446
0447
0448
0449
0450
0451
0452
0453
0454
0455
0456
0457
0458
0459
0460
0461
0462
0463
0464
0465
0466
0467
0468
0469
0470
0471
0472
0473
0474
0475
0476
0477
0478
0479
0480
0481
0482
0483
0484
0485
0486
0487
0488
0489
0490
0491
0492
0493
0494
0495
0496
0497
0498
0499
0500
0501
0502
0503
0504
0505
0506
0507
0508
0509
0510
0511
0512
0513
0514
0515
0516
0517
0518
0519
0520
0521
0522
0523
0524
0525
0526
0527
0528
0529
0530
0531
0532
0533
0534
0535
0536
0537
0538
0539
0540
0541
0542
0543
0544
0545
0546
0547
0548
0549
0550
0551
0552
0553
0554
0555
0556
0557
0558
0559
0560
0561
0562
0563
0564
0565
0566
0567
0568
0569
0570
0571
0572
0573
0574
0575
0576
0577
0578
0579
0580
0581
0582
0583
0584
0585
0586
0587
0588
0589
0590
0591
0592
0593
0594
0595
0596
0597
0598
0599
0600
0601
0602
0603
0604
0605
0606
0607
0608
0609
0610
0611
0612
0613
0614
0615
0616
0617
0618
0619
0620
0621
0622
0623
0624
0625
0626
0627
0628
0629
0630
0631
0632
0633
0634
0635
0636
0637
0638
0639
0640
0641
0642
0643
0644
0645
0646
0647
0648
0649
0650
0651
0652
0653
0654
0655
0656
0657
0658
0659
0660
0661
0662
0663
0664
0665
0666
0667
0668
0669
0670
0671
0672
0673
0674
0675
0676
0677
0678
0679
0680
0681
0682
0683
0684
0685
0686
0687
0688
0689
0690
0691
0692
0693
0694
0695
0696
0697
0698
0699
0700
0701
0702
0703
0704
0705
0706
0707
0708
0709
0710
0711
0712
0713
0714
0715
0716
0717
0718
0719
0720
0721
0722
0723
0724
0725
0726
0727
0728
0729
0730
0731
0732
0733
0734
0735
0736
0737
0738
0739
0740
0741
0742
0743
0744
0745
0746
0747
0748
0749
0750
0751
0752
0753
0754
0755
0756
0757
0758
0759
0760
0761
0762
0763
0764
0765
0766
0767
0768
0769
0770
0771
0772
0773
0774
0775
0776
0777
0778
0779
0780
0781
0782
0783
0784
0785
0786
0787
0788
0789
0790
0791
0792
0793
0794
0795
0796
0797
0798
0799
0800
0801
0802
0803
0804
0805
0806
0807
0808
0809
0810
0811
0812
0813
0814
0815
0816
0817
0818
0819
0820
0821
0822
0823
0824
0825
0826
0827
0828
0829
0830
0831
0832
0833
0834
0835
0836
0837
0838
0839
0840
0841
0842
0843
0844
0845
0846
0847
0848
0849
0850
0851
0852
0853
0854
0855
0856
0857
0858
0859
0860
0861
0862
0863
0864
0865
0866
0867
0868
0869
0870
0871
0872
0873
0874
0875
0876
0877
0878
0879
0880
0881
0882
0883
0884
0885
0886
0887
0888
0889
0890
0891
0892
0893
0894
0895
0896
0897
0898
0899
0900
0901
0902
0903
0904
0905
0906
0907
0908
0909
0910
0911
0912
0913
0914
0915
0916
0917
0918
0919
0920
0921
0922
0923
0924
0925
0926
0927
0928
0929
0930
0931
0932
0933
0934
0935
0936
0937
0938
0939
0940
0941
0942
0943
0944
0945
0946
0947
0948
0949
0950
0951
0952
0953
0954
0955
0956
0957
0958
0959
0960
0961
0962
0963
0964
0965
0966
0967
0968
0969
0970
0971
0972
0973
0974
0975
0976
0977
0978
0979
0980
0981
0982
0983
0984
0985
0986
0987
0988
0989
0990
0991
0992
0993
0994
0995
0996
0997
0998
0999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
198
```

```
Description of panImain
clear AnalyisPanPos nbuttons buttonWidth buttonPos

0059
0060
0061 DataPanPos = [.015 .1633 .975 .615]; % in pixels [10 panelDimens(4)-415 panelDimens(3)-20 305];
0062 hDataPanel = uipanel('Parent',currPanel,'Title','Data Control','BackgroundColor',[1,1,1],'FontSize',guiFontSize-1,'Units','normalized','Position',DataPanPos);
0063 logospace = [48 39]*scaleX;
0064 icon2 = axes('Parent',hDataPanel,'Units','normalized','Position',[.01 0.81 0.3 0.3]);
0065 iconSize = getpixelposition(icon2); iconSize(3:4) = logospace; setpixelposition(icon2,round(iconSize));
0066 image(imread('icondata.jpg'),'Parent',icon2);
0067 axis(icon2,'off');
0068 clear logospace icon2
0069
0070 nbuttons = 3;
0071 buttonWidth = (DataPanPos(3)-.0625-.0125*(nbuttons+1))/nbuttons;
0072 buttonPos = [DataPanPos(1)+.075 .9 buttonWidth .08];
0073 uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Load','TooltipString','Load data from file','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hLoadDataButtonCallback,'Style','pushbutton');
0074 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0075 % uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Create','FontSize',guiFontSize,'Visible','on','Enable','off','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton');
0076 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0077 % uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Edit','FontSize',guiFontSize,'Visible','on','Enable','off','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton');
0078 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0079 % uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Remove','TooltipString','Clear the selected object from memory','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hRemoveButtonCallback,'Style','pushbutton');
0080 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0081 % hExploreButton = uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Explore','TooltipString','Explore the selected objects with the LTPDA
0082 Explorer','FontSize',guiFontSize,'Visible','on','Enable','on','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton');
0083 clear nbuttons buttonWidth buttonPos
0084
0085 nbuttons = 4;
0086 buttonWidth = (DataPanPos(3)-.0625-.0125*(nbuttons+1))/nbuttons;
0087 buttonPos = [DataPanPos(1)+.075 .8 buttonWidth .08];
0088 % hImportButton = uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Import','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hImportButtonCallback,'Style','pushbutton');
0089 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0090 % uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Clear all input','TooltipString','Clear from memory the entire list of input
0091 objects','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@AskConfirm,'clear','Style','pushbutton');
0092 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0093 % uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Export to workspace','TooltipString','Export the selected objects from LTPDA to MATLAB
0094 workspace','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hExportButtonCallback,'Style','pushbutton');
0095 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0096 % uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Save workspace','TooltipString','Save the LTPDA Workspace as .MAT file','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@SaveWorkspaceCallback,'Style','pushbutton');
0097 % buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
0098 % uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Load workspace','TooltipString','Load a workspace from a .MAT file','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@LoadWorkspaceCallback,'Style','pushbutton');
0099 clear nbuttons buttonWidth buttonPos
0100
0101 textPos = [.0125 .87 5 .083];
0102 uicontrol('Parent',hDataPanel,'BackgroundColor',get(hDataPanel,'BackgroundColor'),'HorizontalAlignment','center','Units','normalized','Position',textPos,'String','Input:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
0103 clear textPos
0104
0105 nlists = 2;
0106 listWidth = (DataPanPos(3)-.0125*(nlists+1))/nlists;
0107 listPos = [.0125 .03 listWidth .08];
0108 InputObjList = uicontrol('Parent',hDataPanel,'BackgroundColor',[.05 0.95 0.95],'HorizontalAlignment','center','Units','normalized','Position',listPos,'String','listInput','Value',1,'Max',10,'Min',1,'FontSize',guiFontSize,'FontWeight','normal','Visible','on','Enable','on','Callback',@hListObjCallback,'Tag','InputObjList','Style','list');
0109 % buttonPos(1) = listPos(1)+listWidth+.0125;
0110 % Info panel:
0111 uicontrol('Parent',hDataPanel,'BackgroundColor',[0.94 0.94,1],'Style','listbox','HorizontalAlignment','left','FontSize',guiFontSize,'Units','pixels','Units','normalized','Position',listPos,'Tag','infoPanel');
0112 % hListObjCallback(InputObjList,1);
0113 clear DataPanPos nlists listWidth listPos
0114
0115 hExecutionPanel = uipanel('Parent',currPanel,'Title','Execution control','BackgroundColor',[1,1,1],'FontSize',guiFontSize-1,'Units','normalized','Position',[.0125 .0222 .975 .121]);
0116 logospace = [48 48]*scaleX;
0117 icon3 = axes('Parent',hExecutionPanel,'Units','normalized','Position',[.01 .05 0.3 0.3]);
0118 iconSize = getpixelposition(icon3); iconSize(3:4) = logospace; setpixelposition(icon3,round(iconSize));
0119 image(imread('iconsettings.jpg'),'Parent',icon3);
0120 axis(icon3,'off');
0121 clear logospace icon3
0122
0123 if isempty(getappdata(0,'maintainresults')), setappdata(0,'maintainresults',0); end
0124 % From all blocks at the end of the analysis calculation, 'Foregroundcolor','black','Visible','on','Enable','off','Callback',setappdata(0,'maintainresults',get(gc,'value')), 'Style','checkbox','Tag','enablecheckbox');
0125 if isempty(getappdata(0,'hiddenames')), setappdata(0,'hiddenames',0); end
0126 % work only in the parameters panel', 'ForegroundColor','black','Visible','on','Enable','off','Callback',setappdata(0,'hiddenames',get(gc,'value')), 'Style','checkbox','Tag','enablecheckbox');
0127 if isempty(find_system(bdroot,'SearchDepth',1,'BlockType','SubSystem','Tag','nestedloops')), loopstatus = 0; else loopstatus = 1; end
0128 % the proper panel', 'ForegroundColor','black','Visible','on','Enable','off','Callback',@UpdateNestedLoops,'Style','checkbox','Tag','enablecheckbox');
0129 if runSim == 1, StartExec(); end
0130
0131 %=====  
0132 %=====  
0133 %=====  
0134 %=====  
0135 %=====  
0136 %=====  
0137 %=====  
0138 %=====  
0139 %=====  
0140 %=====  
0141 %=====  
0142 %=====  
0143 %=====  
0144 %=====  
0145 %=====  
0146 %=====  
0147 %=====  
0148 %=====  
0149 %=====  
0150 %=====  
0151 %=====  
0152 %=====  
0153 %=====  
0154 %=====  
0155 %=====  
0156 %=====  
0157 %=====  
0158 %=====  
0159 %=====  
0160 %=====  
0161 %=====  
0162 %=====  
0163 %=====  
0164 %=====  
0165 %=====  
0166 %=====  
0167 %=====  
0168 %=====  
0169 %=====  
0170 %=====  
0171 %=====  
0172 %=====  
0173 %=====  
0174 %=====  
0175 %=====  
0176 %=====  
0177 %=====  
0178 %=====  
0179 %=====  
0180 %=====  
0181 %=====  
0182 %=====  
0183 %=====  
0184 %=====  
0185 %=====  
0186 %=====  
0187 %=====  
0188 %=====  
0189 %=====  
0190 %=====  
0191 %=====  
0192 %=====  
0193 %=====  
0194 %=====  
0195 %=====  
0196 %=====  
0197 %=====  
0198 %=====  
0199 %=====  
0200 %=====  
0201 %=====  
0202 %=====  
0203 %=====  
0204 %=====  
0205 %=====  
0206 %=====  
0207 %=====  
0208 %=====  
0209 %=====  
0210 %=====  
0211 %=====  
0212 %=====  
0213 %=====  
0214 %=====  
0215 %=====  
0216 %=====  
0217 %=====  
0218 %=====  
0219 %=====  
0220 %=====  
0221 %=====  
0222 %=====  
0223 %=====  
0224 %=====  
0225 %=====  
0226 %=====  
0227 %=====  
0228 %=====  
0229 %=====  
0230 %=====  
0231 %=====  
0232 %=====  
0233 %=====  
0234 %=====  
0235 %=====  
0236 %=====  
0237 %=====  
0238 %=====  
0239 %=====  
0240 %=====  
0241 %=====  
0242 %=====  
0243 %=====  
0244 %=====  
0245 %=====  
0246 %=====  
0247 %=====  
0248 %=====  
0249 %=====  
0250 %=====  
0251 %=====  
0252 %=====  
0253 %=====  
0254 %=====  
0255 %=====  
0256 %=====  
0257 %=====  
0258 %=====  
0259 %=====  
0260 %=====  
0261 %=====  
0262 %=====  
0263 %=====  
0264 %=====  
0265 %=====  
0266 %=====  
0267 %=====  
0268 %=====  
0269 %=====  
0270 %=====  
0271 %=====  
0272 %=====  
0273 %=====  
0274 %=====  
0275 %=====  
0276 %=====  
0277 %=====  
0278 %=====  
0279 %=====  
0280 %=====  
0281 %=====  
0282 %=====  
0283 %=====  
0284 %=====  
0285 %=====  
0286 %=====  
0287 %=====  
0288 %=====  
0289 %=====  
0290 %=====  
0291 %=====  
0292 %=====  
0293 %=====  
0294 %=====  
0295 %=====  
0296 %=====  
0297 %=====  
0298 %=====  
0299 %=====  
0300 %=====  
0301 %=====  
0302 %=====  
0303 %=====  
0304 %=====  
0305 %=====  
0306 %=====  
0307 %=====  
0308 %=====  
0309 %=====  
0310 %=====  
0311 %=====  
0312 %=====  
0313 %=====  
0314 %=====  
0315 %=====  
0316 %=====  
0317 %=====  
0318 %=====  
0319 %=====  
0320 %=====  
0321 %=====  
0322 %=====  
0323 %=====  
0324 %=====  
0325 %=====  
0326 %=====  
0327 %=====  
0328 %=====  
0329 %=====  
0330 %=====  
0331 %=====  
0332 %=====  
0333 %=====  
0334 %=====  
0335 %=====  
0336 %=====  
0337 %=====  
0338 %=====  
0339 %=====  
0340 %=====  
0341 %=====  
0342 %=====  
0343 %=====  
0344 %=====  
0345 %=====  
0346 %=====  
0347 %=====  
0348 %=====  
0349 %=====  
0350 %=====  
0351 %=====  
0352 %=====  
0353 %=====  
0354 %=====  
0355 %=====  
0356 %=====  
0357 %=====  
0358 %=====  
0359 %=====  
0360 %=====  
0361 %=====  
0362 %=====  
0363 %=====  
0364 %=====  
0365 %=====  
0366 %=====  
0367 %=====  
0368 %=====  
0369 %=====  
0370 %=====  
0371 %=====  
0372 %=====  
0373 %=====  
0374 %=====  
0375 %=====  
0376 %=====  
0377 %=====  
0378 %=====  
0379 %=====  
0380 %=====  
0381 %=====  
0382 %=====  
0383 %=====  
0384 %=====  
0385 %=====  
0386 %=====  
0387 %=====  
0388 %=====  
0389 %=====  
0390 %=====  
0391 %=====  
0392 %=====  
0393 %=====  
0394 %=====  
0395 %=====  
0396 %=====  
0397 %=====  
0398 %=====  
0399 %=====  
0400 %=====  
0401 %=====  
0402 %=====  
0403 %=====  
0404 %=====  
0405 %=====  
0406 %=====  
0407 %=====  
0408 %=====  
0409 %=====  
0410 %=====  
0411 %=====  
0412 %=====  
0413 %=====  
0414 %=====  
0415 %=====  
0416 %=====  
0417 %=====  
0418 %=====  
0419 %=====  
0420 %=====  
0421 %=====  
0422 %=====  
0423 %=====  
0424 %=====  
0425 %=====  
0426 %=====  
0427 %=====  
0428 %=====  
0429 %=====  
0430 %=====  
0431 %=====  
0432 %=====  
0433 %=====  
0434 %=====  
0435 %=====  
0436 %=====  
0437 %=====  
0438 %=====  
0439 %=====  
0440 %=====  
0441 %=====  
0442 %=====  
0443 %=====  
0444 %=====  
0445 %=====  
0446 %=====  
0447 %=====  
0448 %=====  
0449 %=====  
0450 %=====  
0451 %=====  
0452 %=====  
0453 %=====  
0454 %=====  
0455 %=====  
0456 %=====  
0457 %=====  
0458 %=====  
0459 %=====  
0460 %=====  
0461 %=====  
0462 %=====  
0463 %=====  
0464 %=====  
0465 %=====  
0466 %=====  
0467 %=====  
0468 %=====  
0469 %=====  
0470 %=====  
0471 %=====  
0472 %=====  
0473 %=====  
0474 %=====  
0475 %=====  
0476 %=====  
0477 %=====  
0478 %=====  
0479 %=====  
0480 %=====  
0481 %=====  
0482 %=====  
0483 %=====  
0484 %=====  
0485 %=====  
0486 %=====  
0487 %=====  
0488 %=====  
0489 %=====  
0490 %=====  
0491 %=====  
0492 %=====  
0493 %=====  
0494 %=====  
0495 %=====  
0496 %=====  
0497 %=====  
0498 %=====  
0499 %=====  
0500 %=====  
0501 %=====  
0502 %=====  
0503 %=====  
0504 %=====  
0505 %=====  
0506 %=====  
0507 %=====  
0508 %=====  
0509 %=====  
0510 %=====  
0511 %=====  
0512 %=====  
0513 %=====  
0514 %=====  
0515 %=====  
0516 %=====  
0517 %=====  
0518 %=====  
0519 %=====  
0520 %=====  
0521 %=====  
0522 %=====  
0523 %=====  
0524 %=====  
0525 %=====  
0526 %=====  
0527 %=====  
0528 %=====  
0529 %=====  
0530 %=====  
0531 %=====  
0532 %=====  
0533 %=====  
0534 %=====  
0535 %=====  
0536 %=====  
0537 %=====  
0538 %=====  
0539 %=====  
0540 %=====  
0541 %=====  
0542 %=====  
0543 %=====  
0544 %=====  
0545 %=====  
0546 %=====  
0547 %=====  
0548 %=====  
0549 %=====  
0550 %=====  
0551 %=====  
0552 %=====  
0553 %=====  
0554 %=====  
0555 %=====  
0556 %=====  
0557 %=====  
0558 %=====  
0559 %=====  
0560 %=====  
0561 %=====  
0562 %=====  
0563 %=====  
0564 %=====  
0565 %=====  
0566 %=====  
0567 %=====  
0568 %=====  
0569 %=====  
0570 %=====  
0571 %=====  
0572 %=====  
0573 %=====  
0574 %=====  
0575 %=====  
0576 %=====  
0577 %=====  
0578 %=====  
0579 %=====  
0580 %=====  
0581 %=====  
0582 %=====  
0583 %=====  
0584 %=====  
0585 %=====  
0586 %=====  
0587 %=====  
0588 %=====  
0589 %=====  
0590 %=====  
0591 %=====  
0592 %=====  
0593 %=====  
0594 %=====  
0595 %=====  
0596 %=====  
0597 %=====  
0598 %=====  
0599 %=====  
0600 %=====  
0601 %=====  
0602 %=====  
0603 %=====  
0604 %=====  
0605 %=====  
0606 %=====  
0607 %=====  
0608 %=====  
0609 %=====  
0610 %=====  
0611 %=====  
0612 %=====  
0613 %=====  
0614 %=====  
0615 %=====  
0616 %=====  
0617 %=====  
0618 %=====  
0619 %=====  
0620 %=====  
0621 %=====  
0622 %=====  
0623 %=====  
0624 %=====  
0625 %=====  
0626 %=====  
0627 %=====  
0628 %=====  
0629 %=====  
0630 %=====  
0631 %=====  
0632 %=====  
0633 %=====  
0634 %=====  
0635 %=====  
0636 %=====  
0637 %=====  
0638 %=====  
0639 %=====  
0640 %=====  
0641 %=====  
0642 %=====  
0643 %=====  
0644 %=====  
0645 %=====  
0646 %=====  
0647 %=====  
0648 %=====  
0649 %=====  
0650 %=====  
0651 %=====  
0652 %=====  
0653 %=====  
0654 %=====  
0655 %=====  
0656 %=====  
0657 %=====  
0658 %=====  
0659 %=====  
0660 %=====  
0661 %=====  
0662 %=====  
0663 %=====  
0664 %=====  
0665 %=====  
0666 %=====  
0667 %=====  
0668 %=====  
0669 %=====  
0670 %=====  
0671 %=====  
0672 %=====  
0673 %=====  
0674 %=====  
0675 %=====  
0676 %=====  
0677 %=====  
0678 %=====  
0679 %=====  
0680 %=====  
0681 %=====  
0682 %=====  
0683 %=====  
0684 %=====  
0685 %=====  
0686 %=====  
0687 %=====  
0688 %=====  
0689 %=====  
0690 %=====  
0691 %=====  
0692 %=====  
0693 %=====  
0694 %=====  
0695 %=====  
0696 %=====  
0697 %=====  
0698 %=====  
0699 %=====  
0700 %=====  
0701 %=====  
0702 %=====  
0703 %=====  
0704 %=====  
0705 %=====  
0706 %=====  
0707 %=====  
0708 %=====  
0709 %=====  
0710 %=====  
0711 %=====  
0712 %=====  
0713 %=====  
0714 %=====  
0715 %=====  
0716 %=====  
0717 %=====  
0718 %=====  
0719 %=====  
0720 %=====  
0721 %=====  
0722 %=====  
0723 %=====  
0724 %=====  
0725 %=====  
0726 %=====  
0727 %=====  
0728 %=====  
0729 %=====  
0730 %=====  
0731 %=====  
0732 %=====  
0733 %=====  
0734 %=====  
0735 %=====  
0736 %=====  
0737 %=====  
0738 %=====  
0739 %=====  
0740 %=====  
0741 %=====  
0742 %=====  
0743 %=====  
0744 %=====  
0745 %=====  
0746 %=====  
0747 %=====  
0748 %=====  
0749 %=====  
0750 %=====  
0751 %=====  
0752 %=====  
0753 %=====  
0754 %=====  
0755 %=====  
0756 %=====  
0757 %=====  
0758 %=====  
0759 %=====  
0760 %=====  
0761 %=====  
0762 %=====  
0763 %=====  
0764 %=====  
0765 %=====  
0766 %=====  
0767 %=====  
0768 %=====  
0769 %=====  
0770 %=====  
0771 %=====  
0772 %=====  
0773 %=====  
0774 %=====  
0775 %=====  
0776 %=====  
0777 %=====  
0778 %=====  
0779 %=====  
0780 %=====  
0781 %=====  
0782 %=====  
0783 %=====  
0784 %=====  
0785 %=====  
0786 %=====  
0787 %=====  
0788 %=====  
0789 %=====  
0790 %=====  
0791 %=====  
0792 %=====  
0793 %=====  
0794 %=====  
0795 %=====  
0796 %=====  
0797 %=====  
0798 %=====  
0799 %=====  
0800 %=====  
0801 %=====  
0802 %=====  
0803 %=====  
0804 %=====  
0805 %=====  
0806 %=====  
0807 %=====  
0808 %=====  
0809 %=====  
0810 %=====  
0811 %=====  
0812 %=====  
0813 %=====  
0814 %=====  
0815 %=====  
0816 %=====  
0817 %=====  
0818 %=====  
0819 %=====  
0820 %=====  
0821 %=====  
0822 %=====  
0823 %=====  
0824 %=====  
0825 %=====  
0826 %=====  
0827 %=====  
0828 %=====  
0829 %=====  
0830 %=====  
0831 %=====  
0832 %=====  
0833 %=====  
0834 %=====  
0835 %=====  
0836 %=====  
0837 %=====  
0838 %=====  
0839 %=====  
0840 %=====  
0841 %=====  
0842 %=====  
0843 %=====  
0844 %=====  
0845 %=====  
0846 %=====  
0847 %=====  
0848 %=====  
0849 %=====  
0850 %=====  
0851 %=====  
0852 %=====  
0853 %=====  
0854 %=====  
0855 %=====  
0856 %=====  
0857 %=====  
0858 %=====  
0859 %=====  
0860 %=====  
0861 %=====  
0862 %=====  
0863 %=====  
0864 %=====  
0865 %=====  
0866 %=====  
0867 %=====  
0868 %=====  
0869 %=====  
0870 %=====  
0871 %=====  
0872 %=====  
0873 %=====  
0874 %=====  
0875 %=====  
0876 %=====  
0877 %=====  
0878 %=====  
0879 %=====  
0880 %=====  
0881 %=====  
0882 %=====  
0883 %=====  
0884 %=====  
0885 %=====  
0886 %=====  
0887 %=====  
0888 %=====  
0889 %=====  
0890 %=====  
0891 %=====  
0892 %=====  
0893 %=====  
0894 %=====  
0895 %=====  
0896 %=====  
0897 %=====  
0898 %=====  
0899 %=====  
0900 %=====  
0901 %=====  
0902 %=====  
0903 %=====  
0904 %=====  
0905 %=====  
0906 %=====  
0907 %=====  
0908 %=====  
0909 %=====  
0910 %=====  
0911 %=====  
0912 %=====  
0913 %=====  
0914 %=====  
0915 %=====  
0916 %=====  
0917 %=====  
0918 %=====  
0919 %=====  
0920 %=====  
0921 %=====  
0922 %=====  
0923 %=====  
0924 %=====  
0925 %=====  
0926 %=====  
0927 %=====  
0928 %=====  
0929 %=====  
0930 %=====  
0931 %=====  
0932 %=====  
0933 %=====  
0934 %=====  
0935 %=====  
0936 %=====  
0937 %=====  
0938 %=====  
0939 %=====  
0940 %=====  
0941 %=====  
0942 %=====  
0943 %=====  
0944 %=====  
0945 %=====  
0946 %=====  
0947 %=====  
0948 %=====  
0949 %=====  
0950 %=====  
0951 %=====  
0952 %=====  
0953 %=====  
0954 %=====  
0955 %=====  
0956 %=====  
0957 %=====  
0958 %=====  
0959 %=====  
0960 %=====  
0961 %=====  
0962 %=====  
0963 %=====  
0964 %=====  
0965 %=====  
0966 %=====  
0967 %=====  
0968 %=====  
0969 %=====  
0970 %=====  
0971 %=====  
0972 %=====  
0973 %=====  
0974 %=====  
0975 %=====  
0976 %=====  
0977 %=====  
0978 %=====  
0979 %=====  
0980 %=====  
0981 %=====  
0982 %=====  
0983 %=====  
0984 %=====  
0985 %=====  
0986 %=====  
0987 %=====  
0988 %=====  
0989 %=====  
0990 %=====  
0991 %=====  
0992 %=====  
0993 %=====  
0994 %=====  
0995 %=====  
0996 %=====  
0997 %=====  
0998 %=====  
0999 %=====  
1000 %=====
```

```

0200 % Button No
0201 dimension = position;
0202 logsize = [77 95];
0203 dimension = [127 , dimension(4)-logsize(2)-50 , logsize];
0204 nobut = axes('Parent',confirmFig,'Units','pixels','Position',dimension,'Tag','No');
0205 image(imread('buttonno2.jpg'),'Parent',nobut,'ButtonDownFcn','set(findobj('Tag','ConfirmReset'),'UserData',1); uiresume','Tag','NoButton');
0206 axis(nobut,'off');
0207
0208 uiwait;
0209
0210 if get(confirmFig,'UserData'), delete(confirmFig); return; end
0211
0212 if strcmp(varargin{3},'close')
0213     delete(confirmFig);
0214     disp('Closing LTPDA GUI...');
0215     delete(timerfindall);
0216     clear global LTPDAinvar LTPDAoutvar;
0217     setappdata(0,'ltpda_repo_user',''); setappdata(0,'ltpda_repo_pass','');
0218     closedeq
0219 elseif strcmp(varargin{3},'clear')
0220     delete(confirmFig);
0221     LTPDAinvar=[];
0222     set(InputObjList,'Value',1)
0223     set(findobj('Tag','infoPanel'),'String','')
0224     set_param(find_system(bdroot,'FindAll','on','type','Annotation','Tag','ltpda model'),'UserData','');
0225     UpdateListBox();
0226 end
0227
0228 %-----
0229
0230 %
0231 %
0232 %-----
0233 function hLoadDataButtonCallback(varargin)
0234 % Callback function: run when the user click the "Add data" button
0235
0236 filenames = uipickfiles('REFilter','txt');
0237 if ~isempty(filenames) && ~isnumeric(filenames(1,1))
0238     [x,y]=size(filenames);
0239     for j=1:y
0240         ext = filenames{j}(end-2:end);
0241         if strcmp(ext,'mat')
0242             varlist = whos('file',filenames{j});
0243             for k=1:numel(varlist)
0244                 newObj = load(filenames{j},varlist(k).name); %ok=NASGU>
0245                 newObj = eval(['newObj.',varlist(k).name]);
0246                 if isa(newObj,'ltpda po')
0247                     LTPDAinvar = [LTPDAinvar; {newObj,1}];
0248                     disp(' + 1 Obj added to the global variable LTPDAinvar')
0249                 end
0250             end
0251         else
0252             anobject={aq(filenames{j}),1};
0253             LTPDAinvar=[LTPDAinvar;anobject];
0254             disp(' + 1 AO added to the global variable LTPDAinvar')
0255         end
0256     end
0257     UpdateListBox();
0258 end
0259
0260 %-----
0261
0262 %-----
0263
0264 function hRemoveButtonCallback(varargin)
0265 % Callback function: run when the user click the "Remove data" button
0266
0267 index_selected = get(InputObjList,'Value');
0268 listObjs = get(InputObjList,'String');
0269 if ~strcmp(listObjs,'The memory is empty')
0270     for i=1:numel(index_selected)
0271         objSelect = str2double(strtok(listObjs(index_selected(i),1),','));
0272         LTPDAinvar(objSelect,i) = [];
0273     end
0274     UpdateListBox();
0275 end
0276 set(findobj('Tag','infoPanel'),'String','')
0277
0278 %-----
0279
0280 %-----
0281
0282 function hExportButtonCallback(varargin)
0283 % Callback function: run when the user click the "Export data" button
0284
0285 currSelObjs = get(InputObjList,'Value');
0286 listObjs = get(InputObjList,'String');
0287 if ~strcmp(listObjs,'The memory is empty')
0288     selectedObjs = [];
0289     for i=1:numel(currSelObjs)
0290         objSelect = str2double(strtok(listObjs(currSelObjs(i),1),','));
0291         selectedObjs = [selectedObjs,LTPDAinvar(objSelect,i)];
0292     end
0293     if ~isempty(selectedObjs)
0294         for i=1:numel(selectedObjs)
0295             objName = [class(selectedObjs{i}),',' ,selectedObjs{i}.name];
0296             objName = genvarname(objName,evalin('base','who'));
0297             assignin('base',objName,selectedObjs{i});
0298             disp(['Saved to workspace with name ',objName])
0299         end
0300     end
0301 end
0302
0303 %-----
0304
0305 %-----
0306
0307 function hExploreButtonCallback(varargin)
0308 % Callback function: run when the user click the "Update list" button
0309
0310 currSelObjs = get(InputObjList,'Value');
0311 listObjs = get(InputObjList,'String');
0312 if ~strcmp(listObjs,'The memory is empty')
0313     selectedObjs = [];
0314     for i=1:numel(currSelObjs)
0315         objSelect = str2double(strtok(listObjs(currSelObjs(i),1),','));
0316         selectedObjs = [selectedObjs,LTPDAinvar(objSelect,i)];
0317     end
0318     if ~isempty(selectedObjs)
0319         ltpda_explorer(selectedObjs);
0320     else
0321         ltpda_explorer(LTPDAinvar);
0322     end
0323 end
0324 UpdateListBox();
0325
0326 %-----
0327
0328 %-----
0329
0330 %-----
0331
0332 function SaveWorkspaceCallback(varargin)
0333 % Callback function: run when the user click the "Save all"
0334 % button
0335
0336 try
0337     [FileName,PathName] = uiputfile(''.mat','Save Workspace As');
0338     if ~isa(FileName,'double'), save(strcat(PathName,FileName),LTPDAinvar,LTPDAoutvar); disp('Workspace saved'); end
0339 catch
0340 end
0341
0342 %-----
0343
0344 %-----
0345
0346 function LoadWorkspaceCallback(varargin)
0347 % Callback function: run when the user click the "Load workspace"
0348 % button

```

```

0349     try
0350         [fileName,PathName] = uigetfile('*.mat','Load Workspace');
0351         temp = load(strcat(PathName,FileName),'LTPDAinvar');
0352         LTPDAinvar = [LTPDAinvar ; temp.LTPDAinvar];
0353         temp = load(strcat(PathName,FileName),'LTPDAoutvar');
0354         LTPDAoutvar = [LTPDAoutvar ; temp.LTPDAoutvar];
0355         disp('Workspace loaded')
0356         clear temp
0357         UpdateListBox()
0358     catch
0359     end
0360 end
0361 %-----
0362 %-----
0363 %function UpdateNestedLoops(hObject,varargin)
0364 % Callback function: run when the user click the "Enable nested loops"
0365 % checkbox.
0366
0367 if ~isempty(bdroot) && ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
0368     % it's selected a ltpda simulink analysis diagram.
0369     status = get(hObject,'Value');
0370     nestBlock = find_system(bdroot,'SearchDepth',1,'BlockType','SubSystem','Tag','nestedloops');
0371     try nestBlock = nestBlock(1); catch end
0372
0373     if status % The user enabled nested loops. Create the nested loops block:
0374         if isempty(nestBlock)
0375             paramLoop = cell(1,6);
0376             nestBlock = add_block('ltpda_library/Commonly Used Blocks/Globals',[bdroot.'/Nested loops']);
0377             annotation = find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model');
0378             position = get_param(annotation,'Position');
0379             set_param(nestBlock,'LinkStatus','inactive','Showname','off','MaskDisplay','disp('Nested loops')','Tag','nestedloops','DropShadow','on','Position',[position(1)+20 position(2)-50 position(1)+150 position(2)-30]);
0380             set_param(nestBlock,'UserData',paramLoop,'UserDataPersistent','on')
0381             ltpdagui('redraw',8);
0382         end
0383     else % The user disabled nested loops. Remove the block:
0384         if ~isempty(nestBlock), delete_block(nestBlock); end;
0385         clear paramLoop;
0386     end;
0387
0388 else
0389     disp('*** Warning: the current model is not a valid LTPDA model. Impossible to enable nested loops.');
```

```

0498         set_param(utilsprog.cbcbsh,'Selected','off');
0499         newBlockHandle=add_block('ltpda_library/Input/Object from list', strcat(bdroot,'/A01'),'MakeNameUnique','on');
0500         children=get_param(newBlockHandle,'Blocks');
0501         inputBlock=strcat(get(newBlockHandle,'Path'),'/',get(newBlockHandle,'Name'),'/',children{1});
0502         set_param(inputBlock,'Value',number);
0503         % Modify the block annotation:
0504         objName = LTPDAInvar(str2double(number),1).name;
0505         if numel(objName)>2 && strcmp(objName,'none'), objName = class(LTPDAInvar(str2double(number),1)); end
0506         if numel(objName)>20, objName = [...,LTPDAInvar(str2double(number),1).name(end-20:end)]; end
0507         set_param(newBlockHandle,'AttributesFormatString',[number,' ',objName]);
0508         set_param(newBlockHandle,'Selected','on');
0509     end
0510 end
0511 %-----
0512 %-----
0513 %-----
0514 %-----
0515 %-----
0516 %-----
0517 function hNewButtonCallback(varargin)
0518 % Callback function run when the user click the "New model" button.
0519 % It starts also the timer associated to the continuous selection check
0520 % (this must be changed in order to allow multiple system execution).
0521
0522     hour      = datestr(now,13); % HH:MM:SS
0523     hour(s)   = hour(4);hour(4)=hour(5);hour(5)=hour(7);hour(6)=hour(8);
0524     currentsystem = strcat(getappdata(0,'ltpda_repo_user'),'datestr(now,7),datestr(now,5),datestr(now,11),'_',hour(1:6));
0525     if ~isempty(find_system('Name',currentsystem)), return; end
0526     simulink
0527     new_system(currentsystem)
0528     open_system(currentsystem)
0529
0530     systemHandle = get_param(gcs,'Handle');
0531     evalin('base','load ltpdaSimConfigSet.mat;');
0532     ltpdaSimConfigSet = evalin('base','load ltpdaSimConfigSet;');
0533     attachConfigSetCopy(systemHandle,'ltpdaSimConfigSet');
0534     setActiveConfigSet(systemHandle,'LTPDA');
0535     detachConfigSet(systemHandle,'Configuration');
0536     evalin('base','clear ltpdaSimConfigSet;');
0537
0538     set(systemHandle,'Location',[200 100 1100 600]);
0539     set(systemHandle,'Toolbar','off');
0540
0541     caption = strcat(gcs,'/1');
0542     add_block('built-in/note',caption,'Position',[100 0 400],'UserData',1);
0543     annotationHandle = find_system(gcs,'FindAll','on','type','annotation');
0544     caption1 = ['Analysis designed by ',getappdata(0,'ltpda_repo_user')];
0545     % caption1b = strcat('created:',char(java.lang.System.getProperties.getProperty('user.name')));
0546     caption2 = ['created (UTC): ',char(time)];
0547     caption3 = 'IP: ',char(getHostAddress(java.net.InetAddress.getLocalHost));
0548     caption4 = 'OS: ',computer;
0549     caption5 = 'MATLAB ver: ',getappdata(0,'matlab_version');
0550     caption6 = 'LTPDA ver: ',getappdata(0,'ltpda_version');
0551     caption = strcat(caption2,caption3,caption4,caption5,caption6); %ok<VCAT>
0552     set_param(annotationHandle,'HorizontalAlignment','left','Text',caption,'Tag','ltpda model')
0553     set_param(annotationHandle,'ClickFcn','disp(''You''re not allowed to change this'')')
0554 end
0555 %-----
0556 %-----
0557 %-----
0558 function hSaveCallback(varargin)
0559 % This callback is called whenever the user click the "Save" button to
0560 % store the current model on disk.
0561
0562     if ~isempty(bdroot) && ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
0563         inputBlockHandles = get_param(find_system(bdroot,'AncestorBlock','ltpda_library/Input/Object from list'),'handle');
0564         for i=1:numel(inputBlockHandles)
0565             children = get_param(inputBlockHandles{i},'Blocks');
0566             inputBlock = strcat(get(inputBlockHandles{i},'Path'),'/',get(inputBlockHandles{i},'Name'),'/',children{1});
0567             currIndex(i) = str2double(get_param(inputBlock,'Value'));
0568             if currIndex(i)~=0
0569                 objcommand = string(LTPDAInvar(currIndex(i)));
0570                 set_param(inputBlockHandles{i},'Description',objcommand);
0571             end
0572         end
0573
0574         oldName = bdroot;
0575         if ~isempty(oldName) && ~isempty(find_system(oldName,'FindAll','on','Type','Annotation','Tag','ltpda model'))
0576             try
0577                 try cd(getappdata(0,'DefaultAnalysesDirectory')); catch end
0578                 [fileName,filePath] = uigetfile([oldName,'.mdl'],'Choose destination folder');
0579                 fileName = [filePath,fileName];
0580                 save_system(oldName,fileName);
0581                 try cd(getappdata(0,'DefaultDataDirectory')); catch end
0582             catch
0583             end
0584         end
0585     end
0586 end
0587 %-----
0588 %-----
0589 %-----
0590 %-----
0591 function hOpenCallback(varargin)
0592 % This callback is called whenever the user click the "Open" button, to
0593 % retrieve a previously saved model from disk.
0594
0595     try
0596         try cd(getappdata(0,'DefaultAnalysesDirectory')); catch end
0597         [fileName,filePath] = uigetfile('*.mdl','Choose source file');
0598         try cd(getappdata(0,'DefaultDataDirectory')); catch end
0599         fileName = [filePath,fileName];
0600         open_system(fileName);
0601         simulink
0602
0603         % Load the globals into memory:
0604         globBlock = find_system(bdroot,'BlockType','SubSystem','Tag','globals');
0605         if ~isempty(globBlock), gl = get_param(globBlock{1},'UserData'); end
0606
0607
0608         % Update the indexes inside 'Object from list':
0609         inputBlockHandles = get_param(find_system(gcs,'AncestorBlock','ltpda_library/Input/Object from list'),'handle');
0610         for i=1:numel(inputBlockHandles)
0611             objcommand = get_param(inputBlockHandles{i},'Description');
0612             if ~isempty(objcommand)
0613                 xx = size(LTPDAInvar,1);
0614                 objcommand = [LTPDAInvar{',num2str(xx+1),'',1}=',objcommand',';LTPDAInvar{',num2str(xx+1),'',2}=0'}];
0615                 eval(objcommand);
0616
0617                 % Update the pointer inside the input block and the annotation:
0618                 children = get_param(inputBlockHandles{i},'Blocks');
0619                 inputBlock = strcat(get(inputBlockHandles{i},'Path'),'/',get(inputBlockHandles{i},'Name'),'/',children{1});
0620                 set_param(inputBlock,'Value',num2str(xx+1));
0621                 set_param(inputBlockHandles{i},'AttributesFormatString',num2str(xx+1));
0622             end
0623         end
0624     end
0625     UpdateListBox()
0626 end
0627 %-----
0628 %-----
0629 %-----
0630 %-----
0631 %-----
0632 %-----
0633 %-----
0634 %-----
0635 %-----
0636 function StartExec(varargin)
0637 % This callback is called whenever the user click the "Start" button.
0638
0639     if isempty(bdroot) || isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model')), return; end
0640
0641     disp('=====')
0642     disp('Beginning the analysis')
0643     disp('=====')
0644     stopTimerFindall('Tag','StartButton');
0645     button = findobj('Tag','StartButton');
0646     set(button,'CData',imread('startred.jpg'));

```

```

drawnow
progressBar();
drawnow
currSystem = bdroot;
allBlocks = find_system(currSystem,'LookUnderMasks','all');
for i=2:numel(allBlocks), curr_param(allBlocks{i},'LinkStatus','inactive'); set_param(allBlocks{i},'MaskVariables',''); end
addterms(currSystem)
%
% Check if it's a partial execution:
% if ismember('executionList',evalin('base','who')), partialExecution = 1;
% evalin('base','executionList')
% else partialExecution = 0;
end
% Clear the previous execution history:
if ~partialExecution
    set_param(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'),'UserData','');
end
% Set all objects present at the startup to be maintained:
if size(LTPDAInvar,1)>0, LTPDAInvar(1,2) = 1; end
% Load the globals from the current model:
globBlock = find_system(bdroot,'BlockType','SubSystem','Tag','globals');
if ~isempty(globBlock), gl = get_param(globBlock{i},'UserData'); end
%
% =====
% Execution with nested loops
% =====
if loopstatus==1
    % Example:
    %      n order appearance
    % paramLoop={
    %      1 1 ... x-----
    %      2 2 ... .x-----
    %      3 3 ... ..x-----
    %      4 2 ... .x-----
    %      5 3 ... .x-----
    %      6 1 ... x-----
    %
    % branches=[];
    % [xx,yy]=size(paramLoop);
    % % xx is the total number of parameters to change in loops.
    % for ii=1:xx
    %     if ii==xx || paramLoop(ii+1,2)==paramLoop(ii,2)
    %         branches=[branches,ii];
    %     end
    %
    % % 'branches' contains the ordinal number of those rows which do not have
    % % a children row, ie. the following rows have higher order. In the
    % % example, branches [ 3 , 5 , 6 ];
    %
    % family=[];
    % numberIter=[];
    % for ii=1:numel(branches)
    %     currOrder=paramLoop(branches(ii),2);
    %     family(ii,1) = 0;
    %     numberIter(ii,1) = 1;
    %     for jj=branches(ii):1:1
    %         if paramLoop(jj,2)==currOrder
    %             family(ii,currOrder) = jj;
    %             family(ii,1) = family(ii,1)+1;
    %             numberIter(ii,currOrder+1) = paramLoop(jj,6);
    %             numberIter(ii,1) = numberIter(ii,1)+paramLoop(jj,6);
    %             currOrder = paramLoop(jj,2)-1;
    %         end
    %     end
    % end
    %
    % % 'family' contains all the parents information for each branch: the
    % % first column contains how many parents there are. In the example:
    % % family = { 3 1 2 3
    % %             3 1 4 5
    % %             1 6
    % % }
    % % Similarly, numberIter contains the number of iteration necessary to
    % % complete each param. The first column contains the total for each
    % % branch/row.
    % % numberIter = { 40 2 4 5
    % %                36 2 6 3
    % %                10 10
    % % }
    %
    % for ii=1:numel(branches)
    %     x = family(ii,1); % x = how many params to change in this branch
    %     status=ones(1,x); % status shows the current step in the loop;
    %                         % [2,3,5] means eg. that we've executed the 2nd
    %                         % value for the first param, the 3rd for the
    %                         % second param, and the 5th for the third param.
    %
    %     for iter=1:numberIter(ii,1)
    %         % =====
    %         % Here goes what must be executed
    %         % =====
    %
    %         if x==1
    %             disp(sprintf('==== Current status is: ===== %d',status(1)));
    %         elseif x==2
    %             disp(sprintf('==== Current status is: ===== %d %d',status(1),status(2)));
    %         elseif x==3
    %             disp(sprintf('==== Current status is: ===== %d %d %d',status(1),status(2),status(3)));
    %         end
    %
    %         for nn=1:numel(status)
    %             value=paramLoop(family(ii,nn+1),9)(status(nn));
    %             paramCommand = BuildCommand(value,family(ii,nn+1));
    %             set(paramLoop(family(ii,nn+1),3),'Description',paramCommand);
    %         end
    %     end
    %
    % try
    % =====
    %
    % [ttt,xxx,yyy] = sim(currSystem); %!ok<NAGSU>
    %
    % =====
    %
    % for jj=1:1:1
    %     if status(jj)<numberIter(ii,jj)+1 % then we still have to complete iterations on i-th row
    %         status(jj)=status(jj)+1;
    %         if jj==x % if we've just increased a line which is not the last one, reset the lower lines
    %             for k=jj+1:x
    %                 status(kk)=1;
    %             end
    %             break
    %         end
    %     end
    % end
    %
    % end
    %
    % end
    %
    % =====
    % End of execution with nested loops
    % =====
    close(findobj('Name','LTPDA Progress Bar'))
    %
    % Remove final results (intermediate were removed during analysis, as
    % soon as they became not useful anymore):
    if ~getappdata(0,'maintainResults') % && ~partialExecution
        for i=size(LTPDAInvar,1)-1:1
            if LTPDAInvar(i,2)==0

```

```
0796         LTPDAinvar(i,:)=[];
0797     % Update the execution history:
0798     ltpda_annotation = find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda_model');
0799     exechHistory = get_param(ltpda_annotation,'UserData');
0800     for xx=1:size(exechHistory,1)
0801         for yy=3:size(exechHistory,2)
0802             if exechHistory{xx,yy} == 1, exechHistory{xx,yy} = -1; end
0803             if exechHistory{xx,yy} > 1, exechHistory{xx,yy} = exechHistory{xx,yy}-1; end
0804         end
0805         if exechHistory{xx,1} == 1, exechHistory{xx,1} = -1; end
0806         if exechHistory{xx,1} > 1, exechHistory{xx,1} = exechHistory{xx,1}-1; end
0807     end
0808     set_param(ltpda_annotation,'UserData',exechHistory);
0809
0810     end
0811 end
0812
0813
0814
0815 UpdateListbox();
0816 button = findobj('Tag','StartButton');
0817 set(button,'CData',imread('startgreen.jpg'));
0818 evalin('base','clear executionList');
0819 disp('=====')
0820 disp('    Analysis concluded')
0821 disp('=====')
0822 start(timerfindall)
0823
0824 % =====
0825 catch ME
0826 % =====
0827
0828 % Remove intermediate results:
0829 keepRes = getappdata(0,'maintainresults');
0830 if ~keepRes
0831     for i=size(LTPDAinvar,1):-1:1
0832         if LTPDAinvar(i,2)==0, LTPDAinvar(i,:)=[]; end
0833     end
0834 end
0835
0836 UpdateListbox();
0837 close(findobj('Name','LTPDA Progress Bar'))
0838 button = findobj('Tag','StartButton');
0839 set(button,'CData',imread('startgreen.jpg'));
0840 evalin('base','clear executionList');
0841 start(timerfindall)
0842
0843 rethrow(ME)
0844 end
0845
0846 end
0847 %-----
0848
0849 %
0850
0851 end
```

pan2param

PURPOSE ^

This is the GUI m-file able to read the list of parameters to be set for

SYNOPSIS ^

function pan2param(varargin)

DESCRIPTION ^

This is the GUI m-file able to read the list of parameters to be set for the function/block currently selected and to build the window with the parameters panel accordingly.

\$Id: pan2param.m,v 1.46 2008/08/30 12:27:08 nicola Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

 [uipickfiles](#) uipickfiles: GUI program to select file(s) and/or directories.

This function is called by:

 [ltpdagui](#) =====

SUBFUNCTIONS ^

-  function buildplistFigureParam(varargin)
-  function ContinuousParamCheck(varargin)
-  functionname{ij}=get_param(childpath,
-  functionName = get_param(childpath,
-  functionName = description(funcname+14:i-2);
-  function editNameCallback(hObject, varargin)
-  function editValueCallback(hObject, varargin)
-  function browseCallback(hObject, varargin)
-  function WindListCallback(hObject, varargin)
-  function WindlengthCallback(hObject,varargin)
-  function WindPSLCallback(hObject,varargin)
-  function pzmodelLoad(hObject, varargin)
-  function pzmodelGainEdit(hObject, varargin)
-  function pzmodelPZEdit(hObject,varargin)
-  function poleEdit(hObject, varargin)
-  function zeroEdit(hObject, varargin)
-  function AddParamCallback(hObject,varargin)
-  function RemParamCallback(hObject, varargin)
-  function fromSimulinkCallback(hObject, varargin)
-  function UpdateInports(varargin)
-  function numberDataInput(hObject,varargin)
-  function EnableCallback(hObject,varargin)
-  function ExpandEdit(hObject,varargin)
-  function [array]=readInput(inputdata, maxnumb, varargin)


```
function paramcommand = BuildParamCommand(paramlist,varargin)
function paramcommand = RetrievePlist(varargin)
functionname=get_param(childpath,
functionname = description(funcname+14:i-2);
function AnnotationUpdate(paramlist,varargin)
function functionNameSet(hObject, varargin)
functionName2 = get(hObject,
functionName = functionName2;
functionName2 = strcat(functionName,
functionName = description(funcname+14:i-2);
function SetsUpdate(hObject,varargin)
functionname = getappdata(0,
function HelpButtonCallback(varargin)
functionname = get_param(childpath,
functionname = description(funcname+14:i-2);
function ResetButtonCallback(varargin)
function KeepResultCallback(hObject,varargin)
function SetNameCallback(hObject,varargin)
function ApplyButtonCallback(hObject,varargin)
function UndoButtonCallback(varargin)
function LoadInputParamCallback(varargin)
function InputEditCallback(hObject, varargin)
function InputPopupCallback(hObject, varargin)
function AddButtonCallback(varargin)
function LoadMuxParamCallback(varargin)
function MuxEditCallback(hObject, varargin)
function LoadFromBlockCallback(varargin)
function FindOriginCallback(varargin)
function arithmetic(varargin)
function eqEdit(hObject,varargin)
function ErrorFuncNotSupported(varargin)
function ErrorSelectionInconsistent(varargin)
```

SOURCE CODE ^

```
0001 function pan2param(varargin)
0002 % This is the GUI m-file able to read the list of parameters to be set for
0003 % the function/block currently selected and to build the window with the
0004 % parameters panel accordingly.
0005 %
0006 % $Id: pan2param.m,v 1.46 2008/08/30 12:27:08 nicola Exp $
0007
0008 global params newparams paramEnabled oldparamEnabled paramSets
0009 global LTPDAinvar selBlocks objName noParamsReq functionName guiFontSize
0010
0011 %%
0012
0013 currPanel = varargin{1};
0014 panelDimens = get(currPanel, 'Position');
0015 backColor = get(currPanel, 'BackgroundColor');
0016 selBlocks = 0;
0017
0018 guiFontSize = getappdata(0, 'ltpda_gui_fontsize');
0019 if isempty(guiFontSize), guiFontSize = 10; setappdata(0, 'ltpda_gui_fontsize',10); end
0020 set(gcf,'NextPlot','new');
0021 alltimers = timerfindall;
0022 stop(alltimers(1));
0023 set(alltimers(1),'TimerFcn',@ContinuousParamCheck,'Period',0.3);
0024 start(alltimers(1));
0025
0026
0027 %%
0028 %-----
0029 function buildplistFigureParam(varargin)
0030 %-----
0031 % params = []; paramEnabled = []; functionName = []; paramSets = [];
0032 panelDimens = get(currPanel, 'Position');
```

```

0033 lineSpacing = 30 * guiFontSize/10;
0034 setappdata(0,'ltpda_gui_LineSpacing',lineSpacing);
0035 if ~isempty(findobj(gcf,'Tag','apply'))
0036     notredrawn = 0; undoStatus = get(findobj(gcf,'Tag','undo'),'enable');
0037     % deletes only the dynamical part of the panel:
0038     for i = 1:numel(objName), try delete(objName{i}); catch end; end
0039     delete(findobj('Parent',currPanel,'TooltipString','Remove this parameter'))
0040 else
0041     notredrawn = 1; undoStatus = 'off';
0042     % deletes all, to redraw the entire panel:
0043     delete(findobj(gcf,'Parent',currPanel));
0044 end
0045 objName = [];
0046 close(findobj('Tag','expandedEditField'))
0047 if notredrawn % Executes the STATIC part too only if it's not just an update of the panel
0048 % =====
0049 % =====
0050 % ===== Build STATIC panel =====
0051 % =====
0052 % =====
0053 % =====
0054 % -----
0055 % Retrieve the parameters:
0056 % -----
0057 if isempty(params) && isempty(newparams) % the user has just selected another block
0058     paramcommand = get_param(gcbh,'Description');
0059     noParamsReq=0;
0060     if isempty(paramcommand)
0061         paramcommand = RetrievePlist();
0062         selBlocks = utils.prog.gcbsh;
0063         for kk=1:length(selBlocks)
0064             set_param(selBlocks(kk),'Description',paramcommand);
0065         end
0066     end
0067     eval(paramcommand)
0068     newparams = params;
0069     oldparamEnabled = paramEnabled;
0070 end
0071 if ~exist('paramEnabled','var') && isempty(find(newparams,'Sets')) && noParamsReq==0
0072     paramEnabled = zeros(1,nparams(newparams));
0073     oldparamEnabled = paramEnabled;
0074 end
0075 childpath = find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');
0076 if numel(get_param(childpath,'Tag'))>6 && strcmp(get_param(childpath,'Tag'),'generic')
0077     % Text:
0078     uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[40 panelDimens(4)-40 100
141,'FontSize',guiFontSize,'String','Function name:','TooltipString','Function name','Visible','on','Style','text');
0081     % Edit field:
0082     uicontrol('Parent',currPanel,'Units','pixels','BackgroundColor',backColor,'HorizontalAlignment','center','Position',[150 panelDimens(4)-40 100
161,'FontSize',guiFontSize,'String',functionName,'TooltipString','Name of the function','Visible','on','Callback',@functionNameSet,'Style','edit');
0083     panelDimens = get(currPanel,'Position');
0084     panelDimens(4) = panelDimens(4)-40;
0085 else
0086     childpath = find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');
0087     setappdata(0,'ltpda_currFunctionName',get_param(childpath,'Tag'));
0088 end
0089 y=nparams(newparams);
0090 if y==0 && ~exist('paramSets','var') && noParamsReq==0
0091     paramcommand=[paramcommand,'noParamsReq=1;'];
0092     selBlocks = utils.prog.gcbsh;
0093     for kk=1:length(selBlocks) set_param(selBlocks(kk),'Description',paramcommand); end
0094     buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
0095 end
0096 % -----
0097 % Fixed buttons:
0098 % -----
0099 % Help Button
0100 uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[20 10 45 guiFontSize*16/10],'FontAngle','italic','FontSize',guiFontSize,'String','?
','TooltipString','Open the Help associated to this function','Visible','on','Callback',@HelpButtonCallback,'Style','pushbutton');
0104 % Reset Parameters Button
0105 uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[75 10 75
guiFontSize*16/10],'FontAngle','italic','FontSize',guiFontSize,'String','Reset','Reset','TooltipString','Reset the parameters for this
Block','Visible','on','Callback',@ResetButtonCallback,'Style','pushbutton');
0106 % Keep local result
0107 probe = get_param(gcbh,'MaskHelp');
0108 if ~isempty(probe) && isa(probe,'char') && strcmp(probe,'probe'), keepRes = 1; else keepRes = 0; end
0109 uicontrol('Parent',currPanel,'Units','pixels','BackgroundColor',backColor,'HorizontalAlignment','left','Position',[330 10 150
guiFontSize*16/10],'FontAngle','italic','FontSize',guiFontSize,'String','Keep local result','TooltipString','Do not delete the result of this block at the end of the analysis
calculation','Visible','on','Callback',@KeepResultCallback,'Value',keepRes,'Style','checkbox');
0110 % 'Set name' checkbox:
0111 setName = get(gcbh,'UserData');
0112 if isempty(setName), setName = 0; end
0113 uicontrol('Parent',currPanel,'Units','pixels','BackgroundColor',backColor,'HorizontalAlignment','left','Position',[200 10 110
guiFontSize*16/10],'FontAngle','italic','FontSize',guiFontSize,'String','Set name','TooltipString','Set the signal name equal to the block
name','Visible','on','Callback',@SetNameCallback,'Value',setName,'Style','checkbox');
0114 % Number of data inputs:

```

```
0115 uicontrol('Parent',currPanel,'TooltipString','Set the number of input expected by this
block','BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[500 6 75 guiFontSize*20/10],'String','N of
inputs','FontAngle','italic','FontSize',guiFontSize,'Visible','on','Style','text');
0116 paramFromSimulink = 0;
0117 for ii=1:y
0118     paramkeys{ii} = newparams.params(ii).key;
0119     paramvals{ii} = newparams.params(ii).val;
0120     if isa(paramvals{ii},'char') && numel(paramvals{ii})>2 && strcmp(paramvals{ii},'->'), paramFromSimulink = paramFromSimulink + 1; end
0121 end
0122 inports = find_system(gcf,'SearchDepth',1,'LookUnderMasks','all','BlockType','Inport');
0123 uicontrol('Parent',currPanel,'TooltipString','Set the number of input expected by this block','BackgroundColor',backColor,'Units','pixels','Position',[585 8 40
guiFontSize*20/10],'String',num2str(numel(inports)-paramFromSimulink),'FontSize',guiFontSize,'Visible','on','Enable','on','UserData',numel(inports)-
paramFromSimulink,'Callback',@numberDataInput,'Tag','numbDataInputs','Style','edit');
0124
0125 % The function has different sets of parameters:
0126 if exist('paramSets','var') && ~isempty(paramSets)
0127     sets = find(paramSets,'sets');
0128     currSet = find(paramSets,'currSet');
0129     % Sets text:
0130     uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-300)/2 panelDimens(4)-44 150
20],'String','Possible params sets:','Visible','on','Style','text');
0131     % Sets popup menu:
0132     uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-300)/2+150 panelDimens(4)-40 250
20],'String',sets,'Value',currSet,'Visible','on','UserData',sets,'Callback',@SetsUpdate,'Style','popup');
0133     panelDimens(4) = panelDimens(4)-40;
0134 end
0135
0136 % -----
0137 % Button to add user-defined params:
0138 % -----
0139 paramTypes = {'double','char','specwin','pz','pzmodel','->'};
0140 for kk=1:y, paramTypes(strmatch(class(newparams.params(kk).val),paramTypes,'exact'))=[]; end
0141 % ParamType popup
0142 hparamType = uicontrol('Parent',currPanel,'TooltipString','Add a user-defined parameter','BackgroundColor',backColor,'Units','pixels','Position',[20 40 100
20],'String',paramTypes,'Value',1,'Visible','on','Enable','on','Style','popupmenu');
0143 % Add params button
0144 uicontrol('Parent',currPanel,'TooltipString','Add a user-defined parameter','Units','pixels','Position',[130 40 20 20],'String','+', 'Visible','on','Enable','on','Callback',
@AddParamCallback,'UserData',hparamType,'Style','pushbutton');
0145
0146 % -----
0147 % Buttons to apply or undo:
0148 % -----
0149 %
0150 logosize = [30,30];
0151 dimension = [panelDimens(3)-135 , 35 , logosize];
0152 logo = axes('Parent',currPanel,'Units','pixels','Position',dimension);
0153 image(imread('buttonyes.jpg'),'Parent',logo);
0154 axis(logo,'off');
0155 logosize = [30,30];
0156 dimension = [panelDimens(3)-135 , 3 , logosize];
0157 logo = axes('Parent',currPanel,'Units','pixels','Position',dimension);
0158 image(imread('buttonno.jpg'),'Parent',logo);
0159 axis(logo,'off');
0160 % Apply button
0161 uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDimens(3)-100 40 80
guiFontSize*20/10],'HorizontalAlignment','right','FontSize',guiFontSize,'String','Apply','Visible','on','Enable','off','Callback',
@ApplyButtonCallback,'DeleteFcn',{@ApplyButtonCallback,0},'UserData',hparamType,'Tag','apply','Style','pushbutton');
0162 % Undo button
0163 uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDimens(3)-100 10 80
guiFontSize*20/10],'HorizontalAlignment','right','FontSize',guiFontSize,'String','Undo','Visible','on','Enable','undoStatus','Callback',
@UndoButtonCallback,'UserData',hparamType,'Tag','undo','Style','pushbutton');
0164
0165 paramPagePosition = panelDimens + [3 65+guiFontSize*16/10 -6 -(80+guiFontSize*16/10)];
0166 if nargin == 0, currPage = 1;
0167 else currPage = varargin{3};
0168 end
0169 uipanel('Parent',currPanel,'BackgroundColor',backColor,'FontSize',7,'Units','pixels','Position',paramPagePosition,'Tag','paramsPage','UserData',currPage);
0170
0171 else % The part above was executed only if it's not an update of the panel
0172     y=newparams(newparams);
0173     for pp=1:y
0174         paramkeys{pp} = newparams.params(pp).key;
0175         paramvals{pp} = newparams.params(pp).val;
0176     end
0177     if exist('paramSets','var') && ~isempty(paramSets), panelDimens(4) = panelDimens(4)-40; end
0178     paramPagePosition = panelDimens + [3 65+guiFontSize*16/10 -6 -(80+guiFontSize*16/10)];
0179 end
0180
0181 % =====
0182 % =====
0183 % ===== Build DYNAMIC panel =====
0184 % =====
0185 % =====
0186
0187 % -----
0188 % Cycle to verify whether gaps (= extra lines for complex params, such as
0189 % pzmodels) are necessary or not:
0190 % -----
0191 totalGaps = 0;
0192 paramThisPage = 0;
0193 lastParamGap = 0;
```

```

0194 paramPages = [];
0195 for ii=1:y
0196     paramThisPage = paramThisPage+1;
0197     switch class(paramvals{ii})
0198     case 'pzmodel'
0199         if numel(paramvals{ii}.poles)>1 || numel(paramvals{ii}.zeros)>1
0200             totalGaps = totalGaps+1;
0201             lastParamGap = 1;
0202         end
0203     case 'pz'
0204         if numel(paramvals{ii})>1
0205             totalGaps = totalGaps+1;
0206             lastParamGap = 1;
0207         end
0208     end
0209 % Check if it's necessary to draw multiple parameters pages:
0210 if (paramThisPage+totalGaps)*lineSpacing > paramPagePosition(4)-20 % panelDimens(4)-40-guiFontSize*16/10
0211     paramPages = [paramPages , paramThisPage-1];
0212     if ii==y, paramPages = [paramPages,1]; end
0213     paramThisPage = 1;
0214     totalGaps = lastParamGap;
0215     lastParamGap = 0;
0216 elseif ii==y
0217     paramPages = [paramPages , paramThisPage];
0218 end
0219 currGap = 0;
0220 % The vector 'paramPages' contains how many params are contained into every page.
0221 if numel(paramPages) > 1
0222     set(findobj('Tag','paramsPage'),'Title',[ 'Page number ', num2str(currPage)] );
0223     uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[200 39 100 20],'String','Go to page: ','FontName','Times New
0224 Roman','FontSize',guiFontSize+2,'FontWeight','bold','Visible','on','Style','text');
0225     for ii = 0: numel(paramPages)-1
0226         uicontrol('Parent',currPanel,'Units','pixels','Position',[300+ii*(30) 40 20 20],'String',num2str(ii+1),'Visible','on','Enable','on','Callback',
0227 @buildplistFigureParam,ii+1,'Style','pushbutton');
0228     end
0229 elseif isempty(paramPages)
0230     paramPages = 0;
0231 end
0232 firstParam = 1;
0233 if nargin>1
0234     if varargin{3}>numel(paramPages), varargin{3} = numel(paramPages); end
0235     for ii = 1:varargin{3}-1
0236         firstParam = firstParam + paramPages(ii);
0237     end
0238     lastParam = firstParam + paramPages(varargin{3})-1;
0239 else
0240     lastParam = paramPages(1);
0241 end
0242 currParam = 0;
0243 for ii = firstParam:lastParam
0244     currParam = currParam + 1;
0245     addedPar = 0;
0246 % Checkbox to enable input setting:
0247 objName{ii,1} = uicontrol('Parent',currPanel,'Units','pixels','Position',[10,panelDimens(4)-
lineSpacing*(currParam+1+currGap),14,14],'Value',paramEnabled(ii),'Visible','on','Enable','on','Callback', @EnableCallback,'Tag',paramkeys{ii},'UserData',ii,'Style','checkbox');
0248 % Remove Param Button:
0249 try %#ok<ALIGN>
0250     if strcmpi(paramkeys{ii}(1:7),'addPar_')
0251         uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDimens(3)-30 panelDimens(4)-lineSpacing*(currParam+1+currGap) 20 20],'String','-','TooltipString','Remove this
0252 parameter','Visible','on','Enable','on','UserData',ii,'Callback', @RemParamCallback,'Style','pushbutton');
0253         addedPar = 1;
0254         paramkeys{ii}(1:7) = [];
0255     end
0256 catch end
0257 switch class(paramvals{ii})
0258 % =====
0259 case {'double'}
0260 % =====
0261 % Key text:
0262 objName{ii,2} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[40 panelDimens(4)-
(4+lineSpacing*(currParam+1+currGap)) 70 20*guiFontSize/10],'String','Name:', 'FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
0263 % Key value:
0264 objName{ii,3} = uicontrol('Parent',currPanel,'Units','pixels','Position',[120 panelDimens(4)-lineSpacing*(currParam+1+currGap) 60
guiFontSize*20/10],'String',paramkeys{ii},'FontSize',guiFontSize+1,'Visible','on','Enable','off','UserData',ii,'Callback',@editNameCallback,'Style','edit');
0265 if ~noParamsReq && ~addedPar
0266     set(objName{ii,2},'Visible','off');
0267     set(objName{ii,3},'Style','text','BackgroundColor',backColor,'Position',[50 panelDimens(4)-(4+lineSpacing*(currParam+1+currGap)) 120 20]);
0268 end
0269 % Val text:
0270 objName{ii,4} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[205 panelDimens(4)-
(4+lineSpacing*(currParam+1+currGap)) 40 20*guiFontSize/10],'String','Value:', 'FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
0271 % Val value:
0272 data = mat2str(paramvals{ii});
0273 objName{ii,5} = uicontrol('Parent',currPanel,'Units','pixels','Position',[250 panelDimens(4)-lineSpacing*(currParam+1+currGap) 200
guiFontSize*20/10],'String',data,'FontSize',guiFontSize+1,'Visible','on','Enable','off','UserData',ii,'Callback',@editValueCallback,'Style','edit');
0274 objName{ii,6} = uicontrol('TooltipString','Retrieve from Simulink','String','From Simulink','FontName','Times New

```

```
Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Position', [500 panelDims(4)-
lineSpacing*(currParam+1+currGap) 120 20], 'Max', 1, 'Min', 0, 'Value', 0, 'Visible', 'on', 'Tag', num2str(ii), 'UserData', [], 'Callback', @fromSimulinkCallback, 'Style', 'checkbox');
0277 % Val button to expand the edit field:
0278 objName{ii, 8} = uicontrol('UserData', objName{ii, 5}, 'TooltipString', 'Expand the edit field', 'Parent', currPanel, 'BackgroundColor', [0.7 0.7
0.7], 'HorizontalAlignment', 'left', 'Position', [455 panelDims(4)+3-lineSpacing*(currParam+1+currGap) 13 13], 'Visible', 'on', 'Callback', @ExpandEdit, 'Style', 'pushbutton');
0279
0280 % Enable parameters from Simulink:
0281 if strcmp(paramvals{ii}, '-->')
0282     set(objName{ii, 4}, 'Visible', 'off');
0283     set(objName{ii, 5}, 'Visible', 'off');
0284     set(objName{ii, 6}, 'Value', 1);
0285     set(objName{ii, 8}, 'Visible', 'off');
0286 end
0287
0288 % =====
0289 case {'char'}
0290 % =====
0291 % Key text:
0292 objName{ii, 2} = uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'center', 'Position', [40 panelDims(4)-
(4+lineSpacing*(currParam+1+currGap)) 70 20], 'String', 'Name:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0293 % Key value:
0294 objName{ii, 3} = uicontrol('Parent', currPanel, 'Units', 'pixels', 'Position', [120 panelDims(4)-lineSpacing*(currParam+1+currGap) 60
20], 'String', paramkeys{ii}, 'FontSize', guiFontSize+1, 'Visible', 'on', 'Enable', 'off', 'UserData', ii, 'Callback', @editNameCallback, 'Style', 'edit');
0295 if ~noParamsReq && ~addedPar
0296     set(objName{ii, 2}, 'Visible', 'off');
0297     set(objName{ii, 3}, 'Style', 'text', 'BackgroundColor', backColor, 'Position', [50 panelDims(4)-(4+lineSpacing*(currParam+1+currGap)) 120 20]);
0298 end
0299
0300 % Val text:
0301 objName{ii, 4} = uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Position', [205 panelDims(4)-
(4+lineSpacing*(currParam+1+currGap)) 40 20], 'String', 'Value:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0302 % Val value:
0303 objName{ii, 5} = uicontrol('Parent', currPanel, 'Units', 'pixels', 'Position', [250 panelDims(4)-lineSpacing*(currParam+1+currGap) 200
20], 'String', [' ', paramvals{ii}, ''], 'FontSize', guiFontSize+1, 'Visible', 'on', 'Enable', 'off', 'UserData', ii, 'Callback', @editValueCallback, 'Style', 'edit');
0304 objName{ii, 6} = uicontrol('TooltipString', 'Retrieve from Simulink', 'String', 'From Simulink', 'FontName', 'Times New
Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Position', [500 panelDims(4)-
lineSpacing*(currParam+1+currGap) 120 20], 'Max', 1, 'Min', 0, 'Value', 0, 'Visible', 'on', 'Tag', num2str(ii), 'UserData', [], 'Callback', @fromSimulinkCallback, 'Style', 'checkbox');
0305 % Val button to expand the edit field:
0306 objName{ii, 8} = uicontrol('UserData', objName{ii, 5}, 'TooltipString', 'Expand the edit field', 'Parent', currPanel, 'BackgroundColor', [0.7 0.7
0.7], 'HorizontalAlignment', 'left', 'Position', [455 panelDims(4)+3-lineSpacing*(currParam+1+currGap) 13 13], 'Visible', 'on', 'Callback', @ExpandEdit, 'Style', 'pushbutton');
0307
0308 % Browse button:
0309 if strcmpi(paramkeys{ii}, 'filename')
0310     objName{ii, 7} = uicontrol('Parent', currPanel, 'Units', 'pixels', 'Position', [560 panelDims(4)-lineSpacing*(currParam+1+currGap) 90
20], 'String', 'Browse', 'Visible', 'on', 'Enable', 'off', 'UserData', objName{ii, 5}, 'Callback', @browseCallback, 'Style', 'pushbutton');
0311     set(objName{ii, 5}, 'Position', [250 panelDims(4)-lineSpacing*(currParam+1+currGap) 300 20]);
0312     set(objName{ii, 6}, 'Position', [670 panelDims(4)-lineSpacing*(currParam+1+currGap) 120 20]);
0313     delete(objName{ii, 8});
0314 end
0315
0316 % Enable parameters from Simulink:
0317 if strcmp(paramvals{ii}, '-->')
0318     set(objName{ii, 4}, 'Visible', 'off');
0319     set(objName{ii, 5}, 'Visible', 'off');
0320     set(objName{ii, 7}, 'Visible', 'off');
0321     try set(objName{ii, 8}, 'Visible', 'off'); catch end
0322     set(objName{ii, 6}, 'Value', 1);
0323 end
0324
0325 % =====
0326 case 'specwin'
0327 % =====
0328 selectWind = paramvals{ii}.type;
0329 listWindows = ['Kaiser', specwin.getTypes];
0330 selectWindValue = strcmp(selectWind, listWindows, 'exact');
0331 % Window text:
0332 objName{ii, 2} = uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'center', 'Position', [40 panelDims(4)-lineSpacing*(currParam+1+currGap)-5
60 20], 'String', 'Window:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0333 % Window type popup:
0334 objName{ii, 3} = uicontrol('Parent', currPanel, 'Units', 'pixels', 'Position', [105 panelDims(4)-lineSpacing*(currParam+1+currGap)-2 90
20], 'String', listWindows, 'TooltipString', 'Window names', 'Value', selectWindValue, 'Visible', 'on', 'Enable', 'off', 'Tag', paramkeys{ii}, 'UserData', ii, 'Callback',
@windListCallback, 'Style', 'popupmenu');
0335 % Length text:
0336 objName{ii, 4} = uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Position', [220 panelDims(4)-lineSpacing*(currParam+1+currGap)-5
60 20], 'String', 'Length:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0337 % Length value:
0338 currLength = num2str(length(newparams.params(ii).val.win));
0339 objName{ii, 5} = uicontrol('Parent', currPanel, 'Units', 'pixels', 'Position', [280 panelDims(4)-lineSpacing*(currParam+1+currGap)-2 70
20], 'String', num2str(currLength), 'TooltipString', 'Window Length', 'Visible', 'on', 'Enable', 'off', 'UserData', ii, 'Callback', @windLengthCallback, 'Style', 'edit');
0340 if strcmp(selectWind, 'Kaiser')
0341     % PSL text if Kaiser:
0342     objName{ii, 6} = uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Position', [365 panelDims(4)-
lineSpacing*(currParam+1+currGap)-5 60 20], 'String', 'PSL:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0343     % PSL value if Kaiser:
0344     currPSL = newparams.params(ii).val.psl;
0345     objName{ii, 7} = uicontrol('Parent', currPanel, 'Units', 'pixels', 'Position', [415 panelDims(4)-lineSpacing*(currParam+1+currGap)-2 60
20], 'String', num2str(currPSL), 'TooltipString', 'Peak Sidelobe', 'Visible', 'on', 'Enable', 'off', 'UserData', ii, 'Callback', @windPSLCallback, 'Style', 'edit');
0346 end
0347 % Simulink checkbox:
0348 objName{ii, 8} = uicontrol('TooltipString', 'Retrieve from Simulink', 'String', 'From Simulink', 'FontName', 'Times New
Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Position', [500 panelDims(4)-
```



```
lineSpacing*(currParam+1+currGap) 120 20], 'Max', 1, 'Min', 0, 'Value', 0, 'Visible', 'on', 'Tag', num2str(ii), 'UserData', paramvals{ii}, 'Callback', @fromSimulinkCallback, 'Style', 'checkbox');
0349
0350 % =====
0351 case 'pzmodel'
0352 % =====
0353 % Simulink checkbox:
0354 objName{ii,6} = uicontrol('TooltipString','Retrieve from Simulink','String','From Simulink','FontName','Times New
Roman','FontSize',guiFontSize+1,'FontWeight','normal','Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[500 panelDimens(4)-
lineSpacing*(currParam+1+currGap) 120 20], 'Max', 1, 'Min', 0, 'Value', 0, 'Visible', 'on', 'Tag', num2str(ii), 'UserData', paramvals{ii}, 'Callback', @fromSimulinkCallback, 'Style', 'checkbox');
0355
0356 clear pzObj currPoles poleFreqs poleQs poleFqs currZeros zeroFreqs zeroQs zeroFqs
0357 pzObj = newparams.params(ii).val;
0358 currPoles = pzObj.poles;
0359 currZeros = pzObj.zeros;
0360 % Load button:
0361 objName{ii,2} = uicontrol('Parent',currPanel,'HorizontalAlignment','left','Position',[50 panelDimens(4)-(3+lineSpacing*(currParam+1+currGap)) 50
20], 'String', 'Load', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'UserData', ii, 'Callback', @pzmodelLoad, 'enable', 'off', 'Style', 'pushbutton');
0362 % Gain text:
0363 objName{ii,3} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[120 panelDimens(4)-lineSpacing*(currParam+1+currGap)-5
40 20*guiFontSize/10], 'String', 'Gain:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0364 % Gain value:
0365 currGain = pzObj.gain;
0366 objName{ii,4} = uicontrol('Parent',currPanel,'Units','pixels','Position',[155 panelDimens(4)-(2+lineSpacing*(currParam+1+currGap)) 60
20], 'String', num2str(currGain), 'FontSize', guiFontSize+1, 'Visible', 'on', 'Enable', 'off', 'UserData', ii, 'Callback', @pzmodelGainEdit, 'Style', 'edit');
0367 % Poles text:
0368 objName{ii,5} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[240 panelDimens(4)-lineSpacing*(currParam+1+currGap)-5
40 20], 'String', 'Poles:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0369 % Zeros text:
0370 objName{ii,7} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[355 panelDimens(4)-lineSpacing*(currParam+1+currGap)-5
40 20], 'String', 'Zeros:', 'FontName', 'Times New Roman', 'FontSize', guiFontSize+1, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0371 % Poles value:
0372 for kk=1:numel(currPoles)
0373     poleFreqs(kk,1) = currPoles(kk).f;
0374     poleQs(kk,1) = currPoles(kk).q;
0375     if ~isnan(poleQs(kk,1))
0376         poleFqs{kk,1}=strcat(num2str(poleFreqs(kk,1)), ',', num2str(poleQs(kk,1)));
0377     else
0378         poleFqs{kk,1}=num2str(poleFreqs(kk,1));
0379     end
0380 end
0381 if numel(currPoles)>1 || numel(currZeros)>1
0382     currGap=currGap+1;
0383 end
0384 if isempty(currPoles)
0385     poleFqs = '';
0386 else
0387     poleFqs=char(poleFqs);
0388 end
0389 objName{ii,6} = uicontrol('Parent',currPanel,'Units','pixels','Position',[280 panelDimens(4)-(2+lineSpacing*(currParam+1+currGap))-(lineSpacing-
30)*(max(numel(currPoles),numel(currZeros))>1) 60
(20+(max(numel(currPoles),numel(currZeros))>1)*lineSpacing)*guiFontSize/10], 'Max', 2, 'String', poleFqs, 'FontSize', guiFontSize+1, 'TooltipString', 'Insert ''freq'' or ''freq,q'' on multiple
lines', 'Visible', 'on', 'Enable', 'off', 'UserData', ii, 'Callback', {@pzmodelPZEdit,1}, 'Style', 'edit');
0390 % Zeros value:
0391 for kk=1:numel(currZeros)
0392     zeroFreqs(kk,1) = currZeros(kk).f;
0393     zeroQs(kk,1) = currZeros(kk).q;
0394     if ~isnan(zeroQs(kk,1))
0395         zeroFqs{kk,1}=strcat(num2str(zeroFreqs(kk,1)), ',', num2str(zeroQs(kk,1)));
0396     else
0397         zeroFqs{kk,1}=num2str(zeroFreqs(kk,1));
0398     end
0399 end
0400 if isempty(currZeros)
0401     zeroFqs = '';
0402 else
0403     zeroFqs = char(zeroFqs);
0404 end
0405 objName{ii,8} = uicontrol('Parent',currPanel,'Units','pixels','Position',[395 panelDimens(4)-(2+lineSpacing*(currParam+1+currGap))-(lineSpacing-
30)*(max(numel(currPoles),numel(currZeros))>1) 60
(20+(max(numel(currPoles),numel(currZeros))>1)*lineSpacing)*guiFontSize/10], 'Max', 2, 'String', zeroFqs, 'FontSize', guiFontSize+1, 'TooltipString', 'Insert ''freq'' or ''freq,q'' on multiple
lines', 'Visible', 'on', 'Enable', 'off', 'UserData', ii, 'Callback', {@pzmodelPZEdit,0}, 'Style', 'edit');
0406 % =====
0407 case 'pz'
0408 % =====
0409 % Simulink checkbox:
0410 objName{ii,6} = uicontrol('TooltipString','Retrieve from Simulink','String','From Simulink','FontName','Times New
Roman','FontSize',guiFontSize+1,'FontWeight','normal','Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[500 panelDimens(4)-
lineSpacing*(currParam+1+currGap) 120 20], 'Max', 1, 'Min', 0, 'Value', 0, 'Visible', 'on', 'Tag', num2str(ii), 'UserData', paramvals{ii}, 'Callback', @fromSimulinkCallback, 'Style', 'checkbox');
0411
0412 currPoles = newparams.params(ii).val;
0413 clear poleFreqs poleQs poleFqs
0414 for kk=1:numel(currPoles)
0415     poleFreqs(kk,1) = currPoles(kk).f;
0416     poleQs(kk,1) = currPoles(kk).q;
0417     if isnan(poleQs(kk,1))
0418         poleFqs{kk,1}=num2str(poleFreqs(kk,1));
0419     else
0420         poleFqs{kk,1}=strcat(num2str(poleFreqs(kk,1)), ',', num2str(poleQs(kk,1)));
0421     end
0422 end
0423
```

```

0424     end
0425     poleFQs=char(poleFQs);
0426     % Poles text:
0427     objName{ii,2} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[50 panelDims(4)-
(4*lineSpacing*(currParam+1+currGap)) 120 20],'String','PZ','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Enable','off','Style','text');
0428     if numel(currPoles)>1, currGap=currGap+1; end
0429     % Poles value:
0430     objName{ii,3} = uicontrol('Parent',currPanel,'Units','pixels','Position',[250 panelDims(4)-(2*lineSpacing*(currParam+1+currGap))-(lineSpacing-30)*(numel(currPoles)>1)
100 (20+(numel(currPoles)>1)*lineSpacing)*guiFontSize/10],'String',poleFQs,'Max',2,'FontSize',guiFontSize,'TooltipString','Insert 'freq' or
'freq,q','Visible','on','Enable','on','UserData',ii,'Callback',@poleEdit,'Style','edit');
0431
0432     % =====
0433     case 'cell'
0434     % =====
0435     if isa(paramvals{ii}{1},'char') && strcmp(paramvals{ii}{1}, '-->')
0436     % Key value:
0437     objName{ii,3} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[50 panelDims(4)-5-lineSpacing*(currParam+1+currGap) 120
20],'String',paramkeys{ii},'Visible','on','Enable','off','UserData',ii,'Callback',@editNameCallback,'Style','text');
0438     % Text: from Simulink environment:
0439     objName{ii,4} = uicontrol('TooltipString','Retrieve from Simulink','Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[230
panelDims(4)-6-lineSpacing*(currParam+1+currGap) 200 20],'String','From Simulink environment','FontName','Times New
Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
0440     fromSimulink = 1;
0441     else
0442     % Key text:
0443     objName{ii,2} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[40 panelDims(4)-6-
lineSpacing*(currParam+1+currGap) 70 20],'String','Name:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
0444     % Key value:
0445     objName{ii,3} = uicontrol('Parent',currPanel,'Units','pixels','Position',[120 panelDims(4)-lineSpacing*(currParam+1+currGap) 60
20],'String',paramkeys{ii},'Visible','on','Enable','off','UserData',ii,'Callback',@editNameCallback,'Style','edit');
0446     if ~noParamsReq && ~addedPar
0447     set(objName{ii,2},'Visible','off');
0448     set(objName{ii,3},'Style','text','BackgroundColor',backColor,'Position',[50 panelDims(4)-5-lineSpacing*(currParam+1+currGap) 120 20]);
0449     end
0450
0451     % Val text:
0452     objName{ii,4} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[205 panelDims(4)-6-
lineSpacing*(currParam+1+currGap) 40 20],'String','Value:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
0453     % Val value:
0454     data = utils.prog.cell2str(paramvals{ii});
0455     objName{ii,5} = uicontrol('Parent',currPanel,'Units','pixels','Position',[250 panelDims(4)-lineSpacing*(currParam+1+currGap) 130
20],'String',data,'Visible','on','Enable','on','UserData',ii,'Callback',@editValueCallback,'Style','edit');
0456     fromSimulink = 0;
0457     end
0458
0459     % Simulink checkbox:
0460     objName{ii,6} = uicontrol('TooltipString','Retrieve from Simulink','String','From Simulink','FontName','Times New
Roman','FontSize',guiFontSize+1,'FontWeight','normal','Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[500 panelDims(4)-
lineSpacing*(currParam+1+currGap) 120
20],'Max',1,'Min',0,'Value',fromSimulink,'Visible','on','Tag',num2str(ii),'UserData',paramvals{ii},'Callback',@fromSimulinkCallback,'Style','checkbox');
0461
0462     % =====
0463     otherwise % Other types of param?
0464     % =====
0465     % Key value:
0466     objName{ii,3} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[50 panelDims(4)-5-lineSpacing*(currParam+1+currGap) 120
20],'String',paramkeys{ii},'FontSize',guiFontSize+1,'Visible','on','Enable','off','UserData',ii,'Callback',@editNameCallback,'Style','text');
0467     % Text: input not supported. Please construct by proper block.
0468     objName{ii,4} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[310 panelDims(4)-4-lineSpacing*(currParam+1+currGap)
200 20],'String','Unsupported yet','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
0469     fromSimulink = 0;
0470
0471     % Simulink checkbox:
0472     objName{ii,6} = uicontrol('TooltipString','Retrieve from Simulink','String','From Simulink','FontName','Times New
Roman','FontSize',guiFontSize+1,'FontWeight','normal','Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[500 panelDims(4)-
lineSpacing*(currParam+1+currGap) 120
20],'Max',1,'Min',0,'Value',fromSimulink,'Visible','on','Tag',num2str(ii),'UserData',paramvals{ii},'Callback',@fromSimulinkCallback,'Style','checkbox');
0473
0474     % =====
0475     end % of SWITCH the param type
0476     % Cycle to activate enabled params:
0477     if paramEnabled(ii) || noParamsReq==1
0478     [xx,yy] = size(objName);
0479     jj=2;
0480     while jj<=yy && ~isempty(objName{ii,jj})
0481     try set(objName{ii,jj},'Enable','on'); set(objName{ii,jj},'ForegroundColor',[0 0 0]); catch end
0482     jj=jj+1;
0483     end
0484     end
0485 end % of cycle among params
0486
0487 end % of buildplistFigureParam function
0488
0489 %-----
0490
0491 %===== Selection Check =====
0492 %
0493 %-----
0494 %
0495 function ContinuousParamCheck(varargin)
0496 % This is the function to execute a continuous check on the status

```

```

0497 % of the current selection (system and block).
0498
0499 currSelection = utils.prog.gcbsh;
0500 % Clear the figure if nothing's selected
0501 if isempty(selBlocks) && isempty(currSelection), return; end
0502 if (~isempty(selBlocks) && isempty(currSelection)) || strcmp(get_param(gcs, 'BlockDiagramType'), 'library')
0503     delete(findobj(gcf, 'Parent', currPanel))
0504     selBlocks = [];
0505     uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'HorizontalAlignment', 'center', 'Position', [(panelDimens(4)-300)/2 panelDimens(4)-100-100 300
30], 'String', 'Nothing currently selected', 'Visible', 'on', 'Style', 'text');
0506     return
0507 end
0508
0509 if ~isempty(bdroot) && ~isempty(find_system(bdroot, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda model'))
0510 % it's selected a ltpda simulink analysis diagram.
0511
0512 if ((numel(selBlocks)==1 && selBlocks~=gcbh) || length(selBlocks)~=length(currSelection)) && ~isempty(currSelection)
0513     for i=1:length(currSelection)
0514         set_param(currSelection(i), 'LinkStatus', 'inactive');
0515         currentTags{i}=get_param(currSelection(i), 'Tag');
0516     end
0517
0518 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0519 if numel(currSelection)>1 % then the user has selected multiple blocks
0520 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0521 % Check for consistency: all methods? same class?
0522 for i=1:numel(currentTags)
0523     if ~strcmp(currentTags{i}, currentTags{1})
0524         ErrorSelectionInconsistent();
0525         selBlocks = currSelection;
0526         return;
0527     end
0528 end
0529 % Check for consistency: same method?:
0530 for i=1:numel(currSelection)
0531     childpath = find_system(currSelection(i), 'LookUnderMasks', 'all', 'BlockType', 'M-S-Function');
0532     functionname{i}=get_param(childpath, 'Tag');
0533     if ~strcmp(functionname{i}, functionname{1}) % different functions, inconsistent selection
0534         ErrorSelectionInconsistent();
0535         selBlocks = currSelection;
0536         return;
0537     end
0538 end
0539 end
0540 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0541
0542 delete(findobj(gcf, 'Parent', currPanel))
0543 selBlocks = currSelection;
0544 params = []; newparams = []; paramEnabled = []; oldparamEnabled = []; functionName = ''; paramSets = [];
0545 if getappdata(0, 'hidenames') && strcmp(get(gcbh, 'ShowName'), 'on'), for i=1:length(selBlocks), set_param(selBlocks(i), 'ShowName', 'off'); end;
0546 elseif ~getappdata(0, 'hidenames') && strcmp(get(gcbh, 'ShowName'), 'off'), for i=1:length(selBlocks), set_param(selBlocks(i), 'ShowName', 'on'); end; end;
0547 end
0548
0549 % Get the new functionName:
0550 childpath = find_system(gcbh, 'LookUnderMasks', 'all', 'BlockType', 'M-S-Function');
0551 functionName = get_param(childpath, 'Tag');
0552 if isempty(functionName)
0553     description = get_param(gcbh, 'Description');
0554     funcname = findstr('functionName=', description);
0555     if funcname
0556         i = funcname+14;
0557         while ~strcmp(description(i), '.'), i=i+1; end
0558         functionName = description(funcname+14:i-2);
0559     end
0560 end
0561
0562 switch strtok(get(gcbh, 'Tag'))
0563 % =====
0564 case {'function', 'freepl', 'method'}
0565     buildplistFigureParam()
0566 % =====
0567 case 'arithmetic'
0568     arithmetic();
0569 % =====
0570 case 'input'
0571     LoadInputParamCallback();
0572 % =====
0573 case {'mux', 'demux'}
0574     LoadMuxParamCallback();
0575 % =====
0576 case 'from'
0577     LoadFromBlockCallback();
0578 % =====
0579 end % of switch
0580
0581 end % end of blocks selection
0582 end % end of system selection check
0583
0584 end
0585 %-----
0586

```



```

0587 %% ===== Double functions =====
0588 % =====
0589 %-----
0590 function editNameCallback(hObject, varargin)
0591     currParamIndex = get(hObject, 'UserData');
0592     newName = get(hObject, 'String');
0593     oldName = newparams.params(currParamIndex).key;
0594     addedPar = 0;
0595     try if strcmpi(oldName(1:7), 'addPar_'), addedPar = 1; oldName(1:7)=[]; end; catch end
0596
0597     if strcmp(newparams.params(currParamIndex).val, '-->')
0598         block2change = find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'Name', lower(oldName));
0599         while ~isempty(find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'Name', lower(newName))),
0600             set_param(block2change{1}, 'Name', lower(newName))
0601         end
0602
0603         if addedPar, newName = ['addPar_', newName]; end
0604         setKey(newparams.params(currParamIndex), newName);
0605     % To change the drawing command:
0606     if strcmp(newparams.params(currParamIndex).val, '-->'), UpdateInports(lower(newName), 2); end
0607
0608     set(findobj('Tag', 'apply'), 'Enable', 'on');
0609     set(findobj('Tag', 'undo'), 'Enable', 'on');
0610
0611 end
0612 %-----
0613 %-----
0614 function editValueCallback(hObject, varargin)
0615
0616     expandedEdit = 0;
0617     if numel(get(hObject, 'UserData'))>1
0618         hObject = get(hObject, 'UserData');
0619         orighObject = hObject(2);
0620         hObject = hObject(1);
0621         expandedEdit = 1;
0622     else
0623         orighObject = hObject;
0624     end
0625     currParamIndex = get(hObject, 'UserData');
0626     newValue = get(hObject, 'String');
0627
0628     if isempty(newValue), set(orighObject, 'String', num2str(newparams.params(currParamIndex).val)); return; end
0629     if strcmp(newValue(1), '{'),
0630         paramType = 'char';
0631         newValue(1) = []; newValue(end) = [];
0632         for j = numel(newValue)-1:-1:2
0633             if strcmp(newValue(j), '{') && (~strcmp(newValue(j-1), '{') && ~strcmp(newValue(j+1), '{'))
0634                 newValue = [newValue(1:j), '{', newValue(j+1:end)];
0635             end
0636         end
0637     elseif ~strcmp(newValue(1), '{'),
0638         paramType = 'double';
0639     else
0640         paramType = 'double';
0641     end
0642
0643     switch paramType
0644     case 'double'
0645         try
0646             newValue = evalin('base', newValue);
0647             if isnumeric(newValue)
0648                 set(orighObject, 'String', mat2str(newValue))
0649             else
0650                 newparams = pset(newparams, newparams.params(currParamIndex).key, newValue);
0651                 paramcommand = BuildParamCommand(newparams);
0652                 for kk=1:length(selBlocks)
0653                     set_param(selBlocks(kk), 'Description', paramcommand);
0654                 end
0655                 AnnotationUpdate(newparams);
0656                 buildplistFigureParam([], [], get(findobj('Tag', 'paramsPage'), 'UserData'))
0657                 return
0658             end
0659         catch
0660             oldValue = newparams.params(currParamIndex).val;
0661             if ~isa(oldValue, 'char'), mat2str(oldValue); end
0662             set(orighObject, 'String', oldValue)
0663             return
0664         end
0665     end
0666
0667     if expandedEdit, paramEnabled(currParamIndex) = 1; end
0668     newparams = pset(newparams, newparams.params(currParamIndex).key, newValue);
0669
0670     set(findobj('Tag', 'apply'), 'Enable', 'on');
0671     set(findobj('Tag', 'undo'), 'Enable', 'on');
0672
0673 end
0674 %-----
0675 %-----
0676 function browseCallback(hObject, varargin)
0677

```

```

0678 [meth,clas] = strtok(get_param(gcbh,'Tag'),' ');
0679 categ = 'none';
0680 if strcmp(meth,'method')
0681     clas = strtrim(clas);
0682     infoObj = eval([clas,'.getInfo('',functionName,'')']);
0683     categ = infoObj.mcategory;
0684 end
0685
0686 if strcmp(categ,'Output')
0687     [fileName,filePath] = uiputfile('*.','Choose destination file');
0688 else
0689     [fileName,filePath] = uigetfile('*.','Choose source file');
0690 end
0691
0692 if fileName==0
0693     return
0694 else
0695     [file , ext] = strtok(fileName,'. ');
0696     if strcmp(ext,'. '), fileName(end)=[]; fileName=[fileName,'.dat']; end
0697     fileName = [filePath,fileName];
0698 end
0699 editFieldHandle = get(hObject,'UserData');
0700 set(editFieldHandle,'String',fileName);
0701
0702 newValue = fileName;
0703 newparams = pset(newparams,'FILENAME',newValue);
0704
0705 set(findobj('Tag','apply'),'Enable','on');
0706 set(findobj('Tag','undo'),'Enable','on');
0707
0708 end
0709 %-----
0710 %% ===== Specwin functions =====
0711 % =====
0712 %-----
0713 function WindListCallback(hObject, varargin)
0714     currParamID = get(hObject,'UserData');
0715     currWindValue = get(hObject,'Value');
0716     listWindows = ['Kaiser',specwin.getTypes];
0717     currWindow = listWindows{currWindValue};
0718     oldWindObj = newparams.params(currParamID).val;
0719     oldWindow = oldWindObj.type;
0720     if strcmp(currWindow,oldWindow), return;
0721     else
0722         currWindLength = str2double(get(objName{currParamID,5},'String'));
0723         if strcmp(currWindow,'Kaiser'), windobj = specwin(currWindow,currWindLength,round(oldWindObj.psl1));
0724         else windobj = specwin(currWindow,currWindLength);
0725         end
0726     end
0727
0728     newparams=pset(newparams,newparams.params(currParamID).key,windobj);
0729
0730     set(findobj('Tag','apply'),'Enable','on');
0731     set(findobj('Tag','undo'),'Enable','on');
0732
0733     if strcmp(currWindow,'Kaiser') || strcmp(oldWindow,'Kaiser'), buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData')); end
0734
0735 end
0736 %-----
0737 %-----
0738 function WindlengthCallback(hObject,varargin)
0739 % Callback function: run when the user change the value in the Length
0740 % editable field for a specwin object
0741
0742     currParamID = get(hObject,'UserData');
0743     selectedWind = newparams.params(currParamID).val.type;
0744     currWindLength = str2double(get(hObject,'String'));
0745
0746
0747     if strcmp(selectedWind,'Kaiser')
0748         windobj = specwin(selectedWind,currWindLength, newparams.params(currParamID).val.psl1);
0749     else
0750         windobj = specwin(selectedWind,currWindLength);
0751     end
0752
0753     newparams=pset(newparams,newparams.params(currParamID).key,windobj);
0754
0755     set(findobj('Tag','apply'),'Enable','on');
0756     set(findobj('Tag','undo'),'Enable','on');
0757
0758 end
0759 %-----
0760 %-----
0761 function WindPSLCallback(hObject,varargin)
0762 % Callback function: run when the user change the value in the PSL
0763 % editable field for a specwin object
0764
0765     currParamID = get(hObject,'UserData');
0766     selectedWind = newparams.params(currParamID).val.type;
0767     currWindLength = numel(newparams.params(currParamID).val.win);
0768     currPSL = round(str2double(get(hObject,'String')));

```

```

0769     set(hObject, 'String', num2str(currPSL))
0770
0771     window = specwin(selectedWind, currWindLength, currPSL);
0772     newparams = pset(newparams, newparams.params(currParamID).key, window);
0773
0774     set(findobj('Tag', 'apply'), 'Enable', 'on');
0775     set(findobj('Tag', 'undo'), 'Enable', 'on');
0776
0777     end
0778 %-----
0779
0780 %% ===== Pzmodel functions =====
0781 % =====
0782 %-----
0783 function pzmodelLoad(hObject, varargin)
0784     currParamIndex = get(hObject, 'UserData');
0785
0786     try [FileName, PathName] = uigetfile('*.xml', 'Select the .XML pzmodel file');
0787         FileAddress = strcat(PathName, FileName);
0788         pzObj = pzmodel(FileAddress);
0789         if isempty(pzObj), return; end
0790         newparams = pset(newparams, newparams.params(currParamIndex).key, pzObj);
0791     catch
0792     end
0793
0794     set(findobj('Tag', 'apply'), 'Enable', 'on');
0795     set(findobj('Tag', 'undo'), 'Enable', 'on');
0796 %     buildplistFigureParam([], [], get(findobj('Tag', 'paramsPage'), 'UserData'))
0797
0798 end
0799 %-----
0800 %-----
0801 function pzmodelGainEdit(hObject, varargin)
0802     currParamIndex = get(hObject, 'UserData');
0803     pzGain = str2double(get(hObject, 'String'));
0804
0805     pzObj = newparams.params(currParamIndex);
0806     pzKey = pzObj.key;
0807     pzObj = pzObj.val;
0808     pzObj = pzmodel(pzGain, pzObj.poles, pzObj.zeros);
0809     newparams = pset(newparams, pzKey, pzObj);
0810
0811     set(findobj('Tag', 'apply'), 'Enable', 'on');
0812     set(findobj('Tag', 'undo'), 'Enable', 'on');
0813
0814 end
0815 %-----
0816 %-----
0817 function pzmodelPZEdit(hObject, varargin)
0818     currParamIndex = get(hObject, 'UserData');
0819     poleORzero = varargin{2};
0820     pzes = get(hObject, 'String');
0821     pzes = readinput(pzes, 2);
0822     [xx, yy] = size(pzes);
0823     pzList = pz();
0824     for kk = 1:xx
0825         if (yy>1 && pzes(kk, 2)==0) || yy==1, pzList(kk) = pz(pzes(kk, 1));
0826         else pzList(kk) = pz(pzes(kk, 1), pzes(kk, 2));
0827     end
0828
0829 end
0830
0831 pzObj = newparams.params(currParamIndex);
0832 pzKey = pzObj.key;
0833 pzObj = pzObj.val;
0834 if poleORzero, pzObj = pzmodel(pzObj.gain, pzList, pzObj.zeros);
0835 else pzObj = pzmodel(pzObj.gain, pzObj.poles, pzList);
0836 end
0837 newparams = pset(newparams, pzKey, pzObj);
0838
0839 set(findobj('Tag', 'apply'), 'Enable', 'on');
0840 set(findobj('Tag', 'undo'), 'Enable', 'on');
0841
0842 if numel(pzObj.poles)>1 || numel(pzObj.zeros)>1, buildplistFigureParam([], [], get(findobj('Tag', 'paramsPage'), 'UserData')); end
0843
0844 end
0845 %-----
0846 %% ===== Pole functions =====
0847 % =====
0848 %-----
0849 function poleEdit(hObject, varargin)
0850     currParamIndex = get(hObject, 'UserData');
0851     poles = get(hObject, 'String');
0852     poles = readinput(poles, 2);
0853     [xx, yy] = size(poles);
0854     poleList = pz();
0855     for kk=1:xx
0856         if (yy>1 && poles(kk, 2)==0) || yy==1, poleList(kk) = pz(poles(kk, 1));
0857         else poleList(kk) = pz(poles(kk, 1), poles(kk, 2));
0858     end
0859 end

```

```

0860 newparams = pset(newparams,newparams.params(currParamIndex).key,poleList);
0861
0862 set(findobj('Tag','apply'),'Enable','on');
0863 set(findobj('Tag','undo'),'Enable','on');
0864
0865 end
0866 %-----
0867
0868 %% ===== Zero functions =====
0869 % =====
0870 %-----
0871 function zeroEdit(hObject, varargin)
0872     currParamIndex = get(hObject,'UserData');
0873     zeros = get(hObject,'String');
0874     zeros = readInput(zeros,2);
0875     [xx,yy] = size(zeros);
0876     zeroList = pz();
0877     for kk=1:xx
0878         if (yy>1 && zeros(kk,2)==0) || yy==1 , zeroList(kk)=pz(zeros(kk,1));
0879         else zeroList(kk)=pz(zeros(kk,1),zeros(kk,2));
0880         end
0881     end
0882     newparams = pset(newparams,newparams.params(currParamIndex).key,zeroList);
0883
0884     set(findobj('Tag','apply'),'Enable','on');
0885     set(findobj('Tag','undo'),'Enable','on');
0886
0887 end
0888 %-----
0889
0890 %% ===== Fixed Buttons Callbacks =====
0891 % =====
0892 %-----
0893 function AddParamCallback(hObject,varargin)
0894
0895     hparamType = get(hObject,'UserData');
0896     if isa(hparamType,'char') && strcmp(hparamType,'input')
0897         key = 'input';
0898         while ~isempty(find(newparams,key)), key =[key,'x']; end
0899         newparams = append(newparams, param(key,'-->'));
0900         UpdateInports(key,1);
0901         paramcommand = BuildParamCommand(newparams);
0902         for kk=1:length(selBlocks)
0903             set_param(selBlocks(kk),'Description',paramcommand);
0904         end
0905         arithmetic();
0906     else
0907         paramTypes = get(hparamType,'String');
0908         paramType = paramTypes{get(hparamType,'Value')};
0909
0910         switch paramType
0911             case {'double','double/char'}
0912                 key = 'addPar_name';
0913                 while ~isempty(find(newparams,key))
0914                     key =[key,'x'];
0915                 end
0916                 newparams = append(newparams, param(key,1));
0917             case 'specwin'
0918                 newParam = param('addPar_Win',specwin('Kaiser',1000,200));
0919                 newparams = append(newparams, newParam);
0920             case 'pzmodel'
0921                 pzObj = pzmodel(1,[pz(10,2),pz(100)],pz(1));
0922                 newParam = param('addPar_pzmodel',pzObj);
0923                 newparams = append(newparams, newParam);
0924             case '-->'
0925                 key = 'addPar_name';
0926                 while find(newparams,key)
0927                     key = [key,'x'];
0928                 end
0929                 newparams = append(newparams, param(key,'-->'));
0930                 UpdateInports(key,1)
0931             case 'pz'
0932                 newparams = append(newparams, param('addPar_pz',pz(1)));
0933
0934         end
0935
0936         if nparams(newparams)==1, paramEnabled=1;
0937         else paramEnabled=[paramEnabled,1];
0938         end
0939
0940         set(findobj('Tag','apply'),'Enable','on');
0941         set(findobj('Tag','undo'),'Enable','on');
0942
0943         buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
0944     end
0945 end
0946 %-----
0947
0948 %-----
0949
0950 function RemParamCallback(hObject, varargin)

```

```

0951     currParamIndex = get(hObject,'UserData');
0952
0953     paramKey = newparams.params(currParamIndex).key;
0954     paramVal = newparams.params(currParamIndex).val;
0955     newparams = remove(newparams,paramKey);
0956     if ~strcmp(get_param(gcbh,'Tag'),'arithmetic'), paramEnabled(currParamIndex)=[]; end
0957
0958     if strcmp(paramVal,'-->') % it's a data from Simulink: delete the relative inport block.
0959         UpdateInports(paramKey,-1);
0960     end
0961     delete(hObject)
0962
0963     set(findobj('Tag','apply'),'Enable','on');
0964     set(findobj('Tag','undo'),'Enable','on');
0965
0966     if strcmp(get_param(gcbh,'Tag'),'arithmetic'), arithmetic();
0967     else buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
0968     end
0969 end
0970
0971 %-----
0972
0973 %-----
0974 function fromSimulinkCallback(hObject, varargin)
0975     currParamIndex = str2double(get(hObject,'Tag'));
0976     status = get(hObject,'Value'); % 1 activated, 0 deactivated
0977
0978     paramval = get(hObject,'UserData');
0979     paramKey = newparams.params(currParamIndex).key;
0980     if status %activated
0981         if isempty(paramval)
0982             newparams = pset(newparams,paramKey,{'-->'});
0983         else
0984             newparams = pset(newparams,paramKey,'-->');
0985         end
0986         paramEnabled(currParamIndex) = 1;
0987         UpdateInports(paramKey,1)
0988     else %deactivated
0989         % Retrieve the original param:
0990         [func,currClass] = strtok(get(gcbh,'Tag'));
0991         infoObj = eval([currClass,'.getInfo('',functionName,'')']);
0992
0993         if numel(infoObj.sets) > 1
0994             currSet = find(paramSets,'currSet');
0995             origParams = infoObj.plists(currSet);
0996         else
0997             origParams = infoObj.plists;
0998         end
0999
1000         paramKey = newparams.params(currParamIndex).key;
1001         origPar = find(origParams,paramKey);
1002         if isempty(origPar), origPar = 1; end
1003         newparams = pset(newparams,paramKey,origPar);
1004         paramEnabled(currParamIndex) = 0;
1005         UpdateInports(paramKey,-1)
1006     end
1007
1008     buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
1009 end
1010
1011 %-----
1012
1013 %-----
1014 function UpdateInports(varargin)
1015 % Update the MaskDisplay and the number of inports when the user
1016 % changes the params from Simulink.
1017
1018     paramKey = varargin{1};
1019     if numel(paramKey)>7 && strcmpi(paramKey(1:7),'addPar_'), paramKey(1:7)=[]; end
1020
1021 % List of parameters coming from Simulink:
1022     simKey = {};
1023     for i=1:nparams(newparams)
1024         if isa(newparams.params(i).val,'char') && strcmp(newparams.params(i).val,'-->')
1025             name = newparams.params(i).key;
1026             try if strcmpi(name(1:7),'addPar_'), name(1:7)=[]; end; catch end
1027             simKey{end+1} = name;
1028         elseif isa(newparams.params(i).val,'cell') && isa(newparams.params(i).val{1},'char') && strcmp(newparams.params(i).val{1},'-->')
1029             name = newparams.params(i).key;
1030             try if strcmpi(name(1:7),'addPar_'), name(1:7)=[]; end; catch end
1031             simKey{end+1} = name;
1032         end
1033     end
1034
1035 % =====
1036 % If varargin{2}==2 no block has been added or subtracted, just
1037 % renamed an existing param; the only thing to perform is update the
1038 % drawing command.
1039     if varargin{2}~=2
1040
1041         % Remove ALL inner inport blocks.

```

```

1042 if varargin{2}== -1, simKey2remove = [simKey,{paramKey}]; else simKey2remove= simKey; end
1043 for i=1: numel(simKey2remove)
1044     try %#ok<ALIGN>
1045         block2delete = utils.prog.find_in_models(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'Name', lower(simKey2remove{i}));
1046         if numel(block2delete)>1, block2delete(1)=[]; end
1047         blockLines = get_param(block2delete{1}, 'LineHandles');
1048         if (blockLines.Outport(1)~= -1 && ~isempty(blockLines.Outport(1))), delete_line(blockLines.Outport(1)); end
1049         delete_block(block2delete);
1050     catch end
1051 end
1052
1053 % Add ALL inner inport blocks anew:
1054 for i=1: numel(simKey)
1055     newBlock = add_block('simulink/Sources/In1', [gcb, '/Inport1'], 'MakeNameUnique', 'on');
1056     nameAlreadyPresent = utils.prog.find_in_models(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'Name', lower(simKey{i}));
1057     if ~isempty(nameAlreadyPresent)
1058         for j=1: numel(nameAlreadyPresent)
1059             if ~strcmp(nameAlreadyPresent{j}, gcb),
1060                 set_param(nameAlreadyPresent{j}, 'Name', [lower(simKey{i}), '_fun']);
1061             end
1062         end
1063     end
1064     set(newBlock, 'Name', lower(simKey{i}))
1065 end
1066
1067 % Now update the inner structure of lines and mux.
1068 muxblock = find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'BlockType', 'Mux');
1069 try %#ok<ALIGN>
1070     muxLines = get_param(muxblock{1}, 'LineHandles');
1071     for k=1: numel(muxLines.Inport), if (muxLines.Inport(k)~= -1 && ~isempty(muxLines.Inport(k))), delete_line(muxLines.Inport(k)); end
1072     end
1073     for k=1: numel(muxLines.Outport), if (muxLines.Outport(k)~= -1 && ~isempty(muxLines.Outport(k))), delete_line(muxLines.Outport(k)); end
1074     end
1075 catch end
1076 delete_block(muxblock);
1077 inports = find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'BlockType', 'Inport');
1078 fakeInp = find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'BlockType', 'Ground');
1079 inports = [fakeInp; inports];
1080 if numel(inports)>1
1081     muxblock = add_block('built-in/Mux', [gcb, '/Mux']);
1082     set(muxblock, 'Position', [70, 10, 73, 150]);
1083     set(muxblock, 'Inputs', num2str(numel(inports)))
1084 end
1085 for k=1: numel(inports)
1086     inports{k}=get_param(inports{k}, 'Handle');
1087     blkLines = get(inports{k}, 'LineHandles');
1088     for j=1: numel(blkLines.Outport), if (blkLines.Outport(j)~= -1 && ~isempty(blkLines.Outport(j))), delete_line(blkLines.Outport(j)); end
1089     end
1090     set(inports{k}, 'Position', [10, 10+30*(k-1), 30, 30+30*(k-1)])
1091     set(inports{k}, 'Tag', num2str(k))
1092     if numel(inports)>1
1093         add_line(gcb, [get(inports{k}, 'Name'), '/1'], [get(muxblock, 'Name'), '/1', num2str(k)]);
1094     end
1095 end
1096 funcBlock = find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'BlockType', 'M-S-Function');
1097 funcBlock = get_param(funcBlock, 'Name');
1098 if numel(inports)>1
1099     add_line(gcb, [get(muxblock, 'Name'), '/1'], [funcBlock{1}, '/1']);
1100 else
1101     add_line(gcb, [get(inports{1}, 'Name'), '/1'], [funcBlock{1}, '/1']);
1102 end
1103
1104 end
1105 % =====
1106
1107
1108 % Updating the drawing command for the block Mask:
1109 fakeInp = find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'BlockType', 'Ground');
1110 drawingCom = '';
1111 if numel(simKey)>0 % then there are params 'from simulink' added
1112     if numel(fakeInp)>0 || strcmp(get_param(gcbh, 'Tag'), 'arithmetic'), currInp=1; else currInp=str2double(get(findobj('Tag', 'numbDataInputs'), 'String'))+1; end
1113     for nn=1: numel(simKey)
1114         drawingCom = [drawingCom, 'port_label(''input'',', num2str(currInp), ',', lower(simKey{1}), '','');'];
1115         simKey{1} = [];
1116         currInp = currInp+1;
1117     end
1118 else % then the only inport is the regular one
1119     funcBlock = find_system(gcb, 'SearchDepth', 1, 'LookUnderMasks', 'all', 'BlockType', 'M-S-Function');
1120     funcName = get_param(funcBlock{1}, 'Tag');
1121     if numel(funcName)>6 && strcmp(funcName(1:6), 'ltpda_'), funcName(1:6)=[]; end
1122     drawingCom = strcat('disp(''', lower(funcName), '','');');
1123 end
1124 set(gcbh, 'MaskDisplay', drawingCom)
1125
1126 end
1127 %-----
1128
1129 %-----
1130 function numberDataInput(hObject, varargin)
1131 % Called whenever the user changes the number of data inputs:
1132
1133

```

```

1133 prevNumb = get(hObject,'UserData');
1134 newNumb = str2double(get(hObject,'String'));
1135 if isnan(newNumb), set(hObject,'String',num2str(prevNumb)); return; end
1136
1137 fakeInp = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Ground');
1138 if ~isempty(fakeInp) && newNumb>0
1139     blockLines = get_param(fakeInp{1},'LineHandles');
1140     if (blockLines.Output(1)~-1 && ~isempty(blockLines.Output(1))), delete_line(blockLines.Output(1)); end
1141     delete_block(fakeInp{1})
1142 end
1143
1144 dataInports = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Inport');
1145
1146 paramFromSimulink = 0;
1147 for ii=1:nparams(newparams)
1148     paramvals{ii} = newparams.params(ii).val;
1149     if isa(paramvals{ii},'char') && numel(paramvals{ii})>2 && strcmp(paramvals{ii}, '-->'), paramFromSimulink = paramFromSimulink + 1; end
1150 end
1151
1152 % To set the proper port number for parameters inport:
1153 for ii=1:paramFromSimulink
1154     set_param(dataInports{end-paramFromSimulink+ii}, 'Port', num2str(newNumb+11));
1155 end
1156 % To avoid considering inports for parameters:
1157 dataInports(end-paramFromSimulink+1:end) = [];
1158
1159 % To remove all previous data inports:
1160 for ii=1:numel(dataInports)
1161     try %#ok<ALIGN>
1162         blockLines = get_param(dataInports{ii}, 'LineHandles');
1163         if (blockLines.Output(1)~-1 && ~isempty(blockLines.Output(1))), delete_line(blockLines.Output(1)); end
1164         delete_block(dataInports{ii});
1165     catch end
1166 end
1167
1168 % To add new inports for all data expected:
1169 if newNumb==0, add_block('simulink/Sources/Ground', [gcb, '/data_input'], 'MakeNameUnique','on','Name','data'); end
1170 for ii=1:newNumb
1171     newBlock = add_block('simulink/Sources/In1', [gcb, '/Inport1'], 'MakeNameUnique','on','Name',[ 'data', num2str(ii)]);
1172     set_param(newBlock, 'Port', num2str(ii));
1173 end
1174
1175 UpdateInports('',1)
1176
1177 end
1178 %-----
1179
1180 %-----
1181 function EnableCallback(hObject,varargin)
1182 % Callback function: run when the user click on the checkbox associated
1183 % to each parameter, to enable its setting.
1184
1185 currParamID = get(hObject,'UserData');
1186 [xx,yy] = size(objName);
1187 switch get(hObject,'Value')
1188     case 1 % enable parameter change
1189         jj=2;
1190         while jj<=yy && ~isempty(objName{currParamID,jj})
1191             try set(objName{currParamID,jj}, 'Enable','on'); set(objName{currParamID,jj}, 'ForegroundColor',[0 0 0]); catch end
1192             jj=jj+1;
1193         end
1194         paramEnabled(currParamID)=1;
1195     case 0 % disable parameter change
1196         jj=2;
1197         while jj<=yy && ~isempty(objName{currParamID,jj})
1198             try %#ok<ALIGN>
1199                 if strcmp(get(objName{currParamID,jj}, 'Style'), 'checkbox') && strcmp(get(objName{currParamID,jj}, 'String'), 'From Simulink')
1200                     else
1201                         set(objName{currParamID,jj}, 'Enable','off');
1202                         set(objName{currParamID,jj}, 'ForegroundColor',[0.7 0.7 0.7]);
1203                     end
1204                 catch end
1205                 jj=jj+1;
1206             end
1207         end
1208         paramEnabled(currParamID)=0;
1209     end
1210
1211 set(findobj('Tag','apply'),'Enable','on');
1212 set(findobj('Tag','undo'),'Enable','on');
1213
1214 end
1215 %-----
1216
1217 %-----
1218 function ExpandEdit(hObject,varargin)
1219 % Callback function: run when the user click on the checkbox associated
1220 % to each parameter, to enable its setting.
1221
1222 origEditField = get(hObject,'UserData');
1223 currParamIndex = get(origEditField,'UserData');
1224 paramkey = newparams.params(currParamIndex).key;

```



```

1224     if numel(paramkey)>6 && strcmpi(paramkey(1:7), 'addpar_'), paramkey(1:7)=[]; end
1225     paramval = get(origEditField, 'String');
1226
1227     editFigPos = [200,200,800,300];
1228
1229     editFigure =
figure('Name', paramkey, 'Tag', 'expandedEditField', 'MenuBar', 'none', 'ToolBar', 'none', 'NumberTitle', 'off', 'Units', 'pixels', 'Position', editFigPos, 'Resize', 'on', 'ResizeFcn', 'refresh(gcf)', 'Color',
    , get(0, 'defaultuicontrolbackgroundcolor'));
1230     % Edit field title:
1231     uicontrol('Parent', editFigure, 'HorizontalAlignment', 'left', 'Position', [5 editFigPos(4)-30 editFigPos(3)-10 20], 'String', 'Use this box to type in the value of the selected
parameter:', 'FontName', 'Times New Roman', 'FontSize', 9, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
1232     % Edit field:
1233     editField = uicontrol('Parent', editFigure, 'String', paramval, 'UserData', currParamIndex, 'Units', 'pixels', 'HorizontalAlignment', 'left', 'Position', [5 30 editFigPos(3)-7
editFigPos(4)-60], 'FontSize', guiFontSize+1, 'Visible', 'on', 'max', 100, 'Style', 'edit');
1234     % Ok button:
1235     uicontrol('UserData', [editField, origEditField], 'TooltipString', 'Accept this setting', 'String', 'Ok', 'Parent', editFigure, 'BackgroundColor', [0.7 0.7
0.7], 'HorizontalAlignment', 'center', 'Position', [editFigPos(3)-150 5 70 20], 'Visible', 'on', 'Callback', @editValueCallback, 'Style', 'pushbutton');
1236     % Cancel button:
1237     uicontrol('TooltipString', 'Discard this setting', 'String', 'Cancel', 'Parent', editFigure, 'BackgroundColor', [0.7 0.7
0.7], 'HorizontalAlignment', 'center', 'Position', [editFigPos(3)-75 5 70 20], 'Visible', 'on', 'Callback', 'closereq', 'Style', 'pushbutton');
1238     end
1239     %-----
1240
1241     %% ===== General subfunctions =====
1242     % =====
1243     %-----
1244     function [array]=readInput(inputdata, maxnumb, varargin)
1245         [xx,yy]=size(inputdata);
1246         % xx is the number of rows, yy is the number of characters in the longest row
1247         % array={};
1248         array=[];
1249         for i=1:xx
1250             rowpos=1;
1251             elemidx(i)=1;
1252             for j=1:yy+1
1253                 if (j==yy+1 || inputdata(i,j)==' ' || strcmp(inputdata(i,j), ' ') ) && (j==1 || (j>1 && (~strcmp(inputdata(i,j-1), ' '))))
1254                     if (j==1 || strcmp(inputdata(i,j-1), ' ')) && strcmp(inputdata(i,j), ' ')
1255                         array(i,elemidx(i))=0;
1256                         elemidx(i)=elemidx(i)+1;
1257                         rowpos=j+1;
1258                     else
1259                         % To consider numbers, eg [1]:
1260                         % try array{i,elemidx(i)}=eval(inputdata(i,rowpos:j-1)); catch elemidx(i)=elemidx(i)-1; end
1261                         % To consider strings, eg '1':
1262                         % array{i,elemidx(i)}=inputdata(i,rowpos:j-1);
1263                         array(i,elemidx(i))=eval(inputdata(i,rowpos:j-1));
1264                         rowpos=j+1;
1265                         elemidx(i)=elemidx(i)+1;
1266                     end
1267                 elseif j<yy+1 && strcmp(inputdata(i,j), ' ')
1268                     rowpos=j+1;
1269                 end
1270             if maxnumb==0
1271                 if elemidx(i)==maxnumb+1, break; end
1272             end
1273             end
1274             elemidx(i)=elemidx(i)-1;
1275         end
1276     end
1277     %-----
1278     %-----
1279     function paramcommand = BuildParamCommand(paramlist, varargin)
1280     % Reconstruct the command line necessary to build a given plist
1281
1282     % Initialize of the paramcommand object:
1283     paramcommand = strcat('params=', string(paramlist), ',');
1284
1285     if strcmpi(get_param(gcbb, 'Tag'), 'arithmetic'), return; end
1286
1287     if exist('paramEnabled', 'var') && ~isempty(paramEnabled)
1288         paramcommand = [paramcommand, ' paramEnabled=', mat2str(paramEnabled), ','];
1289     end
1290
1291     if ~exist('newparams', 'var') || nparams(paramlist)==0, paramcommand='params=plist(); paramEnabled=[];'; end
1292
1293     if exist('noParamsReq', 'var') && noParamsReq==1
1294         paramcommand = [paramcommand, ' noParamsReq=1;'];
1295     end
1296
1297     description = get_param(gcbb, 'Description');
1298     funcname = findstr('functionName=', description);
1299     if funcname
1300         i = funcname+14;
1301         while ~strcmp(description(i), ';'), i=i+1; end
1302         funcname = description(funcname+13:i-1);
1303         paramcommand = [paramcommand, ' functionName=', funcname, ','];
1304     end
1305
1306     if exist('paramSets', 'var') && ~isempty(paramSets)
1307         paramcommand = [paramcommand, ' paramSets=', string(paramSets), ','];
1308     end

```



```

1309     end
1310
1311 end
1312 %-----
1313
1314 %-----
1315 function paramcommand = RetrievePlist(varargin)
1316 % Retrieve the name of the function from the block, then asks for
1317 % parameters.
1318
1319 try functionname = varargin{1};
1320 catch
1321     childpath = find_system(gcbh, 'LookUnderMasks', 'all', 'BlockType', 'M-S-Function');
1322     functionname=get_param(childpath, 'Tag');
1323 end
1324
1325 if isempty(functionname)
1326     description = get_param(gcbh, 'Description');
1327     funcname = findstr('functionName=', description);
1328     if funcname
1329         i = funcname+14;
1330         while ~strcmp(description(i), ';'), i=i+1; end
1331         functionname = description(funcname+14:i-2);
1332     end
1333 end
1334
1335 setappdata(0, 'ltpda_currFunctionName', functionname);
1336
1337 try
1338     switch functionname
1339     case 'generic'
1340         paramcommand = 'functionName='generic';params=plist();paramEnabled=[];';
1341         return
1342     case 'arithmetic'
1343         paramcommand = 'functionName='arithmetic';params=plist('alpha', '-->', 'beta', '-->');';
1344         return
1345     otherwise
1346         [func, currClass] = strtok(get(gcbh, 'Tag'));
1347         currClass(1)=[];
1348         infoObj = eval([currClass, '.getInfo(''', functionname, ''')']);
1349
1350         if numel(infoObj.sets) > 1
1351             paramSets = plist('sets', infoObj.sets, 'currSet', 1);
1352             paramcommand = ['paramSets=', string(paramSets), ', ', 'params=plist();'];
1353         elseif numel(infoObj.sets) == 1 && nparams(infoObj.plists)
1354             paramEnabled = zeros(1, nparams(infoObj.plists));
1355             paramcommand = ['params=', string(infoObj.plists), ', ', 'paramEnabled=', mat2str(paramEnabled), ','];
1356         else
1357             paramcommand = 'params=plist();';
1358         end
1359     end
1360 catch ME
1361     disp(sprintf('Error: unable to retrieve the parameters required by the function "%s". Maybe it's not supported yet?', functionname));
1362     ErrorFuncNotSupported();
1363     rethrow(ME)
1364 end
1365
1366 end
1367 %-----
1368
1369 %-----
1370
1371 function AnnotationUpdate(paramlist, varargin)
1372 % Update the block annotation (the text below the block) with all the
1373 % current parameters
1374
1375 annotation = '';
1376 numparams = nparams(paramlist);
1377 firstpar = 1;
1378
1379 if strcmpi(get_param(gcbh, 'Tag'), 'arithmetic')
1380     eq = get(findobj('Tag', 'equationField'), 'String');
1381     if ~isempty(eq)
1382         if numel(eq)>40, eq(1:end-25) = []; eq = ['...', eq]; end
1383         for nn=1:length(selBlocks)
1384             set_param(selBlocks(nn), 'AttributesFormatString', eq);
1385         end
1386     end
1387     return
1388 end
1389
1390 for kk=1:numparams
1391     if paramEnabled(kk)
1392         paramkey = paramlist.params(kk).key;
1393         try %#ok<ALIGN>
1394             if strcmpi(paramkey(1:7), 'addPar_'), paramkey(1:7)=[]; end
1395         catch end
1396         paramval = paramlist.params(kk).val;
1397         switch class(paramval)
1398         case 'specwin'
1399             windowName = paramval.type;

```

```

1400     windowlength = size(paramval.win,2);
1401     if firstpar==1 % it's the first parameter
1402         annotation = ['(specwin) ',windowName, ',',num2str(windowlength)];
1403         firstpar=2;
1404     else % it's not the first parameter
1405         annotation = [annotation,'\n','(specwin) ',windowName, ',',num2str(windowlength)];
1406     end
1407     if strcmp(windowName,'Kaiser')
1408         windowPsl = paramval.psl;
1409         annotation = [annotation,',',num2str(windowPsl)];
1410     end
1411     case 'double'
1412         paramval = mat2str(paramval);
1413         if numel(paramval)>25, paramval=['...',paramval(end-25:end)]; end
1414         if firstpar==1, annotation=[lower(paramkey),', ',paramval]; firstpar=2;
1415         else annotation = [annotation,'\n',lower(paramkey),', ',num2str(paramval)];
1416         end
1417     case 'char'
1418         if numel(paramval)>25, paramval=['...',paramval(end-25:end)]; end
1419         paramval = strrep(paramval,',',',');
1420         if firstpar==1, annotation=[lower(paramkey),', ',paramval]; firstpar=2;
1421         else annotation = [annotation,'\n',lower(paramkey),', ',paramval];
1422         end
1423     case 'pzmodel'
1424         if firstpar==1, annotation='(pzmodel)'; firstpar=2;
1425         else annotation = [annotation,'\n','(pzmodel)'];
1426         end
1427     case 'cell'
1428         if isa(paramval{1},'char') && strcmp(paramval{1},('-->')), strParKey = '-->';
1429         else strParKey = utils.prog.cell2str(paramval);
1430         end
1431         if firstpar==1
1432             annotation = [lower(paramkey),', ',strParKey]; firstpar=2;
1433         else annotation = [annotation,'\n',lower(paramkey),', ',strParKey];
1434         end
1435     case 'pz'
1436         for mm=1:numel(paramval)
1437             poleFreqs(mm,1) = paramval(mm).f;
1438             poleQs(mm,1) = paramval(mm).q;
1439             if ~isnan(poleQs(mm,1))
1440                 poleFqs{mm,1}=strcat(num2str(poleFreqs(mm,1)),', ',num2str(poleQs(mm,1)));
1441             else
1442                 poleFqs{mm,1}=num2str(poleFreqs(mm,1));
1443             end
1444         end
1445         poleFqs=char(poleFqs);
1446         if firstpar==1
1447             annotation = ['pz ',poleFqs(1,:)]; firstpar=2;
1448             for j=2:size(poleFqs,1), annotation = [annotation,'\n','(pz) ',poleFqs(j,:)]; end
1449         else
1450             annotation = [annotation,'\n','pole ',poleFqs(1,:)];
1451             for j=2:size(poleFqs,1), annotation = [annotation,'\n','(pz) ',poleFqs(j,:)]; end
1452         end
1453     case 'zero'
1454         for mm=1:numel(paramval)
1455             zeroFreqs(mm,1) = paramval(mm).f;
1456             zeroQs(mm,1) = paramval(mm).q;
1457             if ~isnan(zeroQs(mm,1))
1458                 zeroFqs{mm,1}=strcat(num2str(zeroFreqs(mm,1)),', ',num2str(zeroQs(mm,1)));
1459             else
1460                 zeroFqs{mm,1}=num2str(zeroFreqs(mm,1));
1461             end
1462         end
1463         zeroFqs=char(zeroFqs);
1464         if firstpar==1
1465             annotation = ['zero ',zeroFqs(1,:)]; firstpar=2;
1466             for j=2:size(zeroFqs,1), annotation = [annotation,'\n','(zero) ',zeroFqs(j,:)]; end
1467         else
1468             annotation = [annotation,'\n','zero ',zeroFqs(1,:)];
1469             for j=2:size(zeroFqs,1), annotation = [annotation,'\n','(zero) ',zeroFqs(j,:)]; end
1470         end
1471     end
1472 end
1473 end
1474 % Finally set the block annotation:
1475 for nn=1:length(selBlocks), set_param(selBlocks(nn),'AttributesFormatString',annotation); end
1476 end
1477 %-----
1478 %-----
1479 function functionNameSet(hObject, varargin)
1480 % Setting the name of the function in the 'generic' block.
1481
1482
1483     functionName2 = get(hObject,'String');
1484     classes = utils.helper.ltpda_userclasses;
1485     for i=1:numel(classes)
1486         if ismethod(eval(classes{i}),functionName2)
1487             currSel = utils.prog.gcbs;
1488             for j=1:numel(currSel), set(currSel(j),'Tag',{'method ',classes{i}}); end
1489             break
1490         else

```

```

1491         currSel = utils.prog.gcbsh;
1492         for j=1:numel(currSel), set(currSel(j), 'Tag', 'function'); end
1493     end
1494 end
1495 paramcommand = RetrievePlist(functionName2);
1496
1497 if ~isempty(paramcommand)
1498     functionName = functionName2;
1499     % setappdata(0, 'ltpda_currFunctionName', functionName);
1500     paramcommand = strcat(paramcommand, 'functionName=', functionName2, ' ');
1501     for i=1:1000
1502         try
1503             for kk=1:length(selBlocks), set_param(selBlocks(kk), 'Name', functionName2); end
1504             break;
1505         catch
1506             functionName2 = strcat(functionName, '_', num2str(i));
1507         end
1508     end
1509     for kk=1:length(selBlocks), set_param(selBlocks(kk), 'Description', paramcommand); end
1510     delete(findobj(gcf, 'Parent', currPanel))
1511     buildplistFigureParam([], [], get(findobj('Tag', 'paramsPage'), 'UserData'))
1512
1513 else % the functionName inserted it's not valid
1514     description = get_param(gcbh, 'Description');
1515     funcname = findstr('functionName=', description);
1516     if funcname
1517         i = funcname+14;
1518         while ~strcmp(description(i, ';'), ';'), i=i+1; end
1519         functionName = description(funcname+14:i-2);
1520     end
1521     set(hObject, 'String', functionName);
1522     % setappdata(0, 'ltpda_currFunctionName', functionName);
1523 end
1524
1525 end
1526
1527 %-----
1528
1529 %% ===== Sets =====
1530 % =====
1531 %-----
1532 function SetsUpdate(hObject, varargin)
1533 % Whenever the user select a different set of params
1534 currSet = get(hObject, 'Value');
1535 paramSets = pset(paramSets, 'currSet', currSet);
1536 functionname = getappdata(0, 'ltpda_currFunctionName');
1537
1538 [func, currClass] = strtok(get(gcbh, 'Tag'));
1539 infoObj = eval([currClass, '.getInfo('', functionname, '')']);
1540
1541 newparams = infoObj.plists(currSet);
1542 paramEnabled = zeros(1, nparams(newparams));
1543
1544 paramcommand = BuildParamCommand(newparams);
1545 for kk=1:length(selBlocks), set_param(selBlocks(kk), 'Description', paramcommand); end
1546
1547 % Finally set the block annotation:
1548 for nn=1:length(selBlocks), set_param(selBlocks(nn), 'AttributesFormatString', ''); end
1549
1550 buildplistFigureParam([], [], get(findobj('Tag', 'paramsPage'), 'UserData'));
1551 end
1552 %-----
1553
1554 %% ===== Special buttons: Help, Reset, KeepResults, SetName, =====
1555 % ===== Apply, Undo =====
1556 %-----
1557 function HelpButtonCallback(varargin)
1558 % Whenever the user click the Help button
1559
1560 try
1561     childpath = find_system(gcbh, 'LookUnderMasks', 'all', 'BlockType', 'M-S-Function');
1562     functionname = get_param(childpath, 'Tag');
1563
1564     description = get_param(gcbh, 'Description');
1565     funcname = findstr('functionName=', description);
1566     if funcname
1567         i = funcname+14;
1568         while ~strcmp(description(i, ';'), ';'), i=i+1; end
1569         functionname = description(funcname+14:i-2);
1570     end
1571
1572 [func, classes] = strtok(get(gcbh, 'Tag'));
1573 if strcmp(func, 'method')
1574     helptext = help([strtrim(classes), '/', functionname]);
1575 else
1576     helptext = help(functionname);
1577 end
1578
1579 helpFigure = figure('MenuBar', 'none', 'ToolBar', 'none', 'Name', strcat(functionname, '
1580 help'), 'NumberTitle', 'off', 'Units', 'pixels', 'Position', [200, 200, 500, 300], 'Resize', 'on', 'ResizeFcn', 'refresh(gcf)', 'Color', get(0, 'defaultuicontrolbackgroundcolor'));
1581 % Help text:

```

```

1581         uicontrol('Parent',helpFigure,'Units','normalized','HorizontalAlignment','left','Position',[0 0 1
1582 1],'FontSize',guiFontSize+1,'String',helptext,'Visible','on','max',100,'Style','edit');
1583     catch
1584         disp('Something's wrong with the creation of the Help figure. Please check 'HelpButtonCallback'')
1585     end
1586 %-----
1587
1588 %-----
1589 function ResetButtonCallback(varargin)
1590 % Whenever the user click the Reset Parameters button
1591
1592 % Ask for confirmation:
1593 screenSize = get(0,'ScreenSize');
1594 backColor = [1 1 1];
1595 position = [(screenSize(3)-250)/2,(screenSize(4)-160)/2,250,160];
1596 confirmFig = figure('Position',position,'Name','Please confirm reset','Tag','ConfirmReset','Resize','off','NumberTitle','off','ToolBar','none','Menubar','none');
1597 % Text
1598 uicontrol('Parent',confirmFig,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[0 position(4)-40 position(3) 20],'String','Are you sure?
1599','FontName','Times New Roman','FontSize',13,'FontWeight','normal','Visible','on','Style','text');
1600 % Button Yes
1601 dimension = position;
1602 logospace = [77 95];
1603 dimension = [50, dimension(4)-logospace(2)-50, logospace];
1604 yesbut = axes('Parent',confirmFig,'Units','pixels','Position',dimension,'Tag','Yes');
1605 image(imread('buttonyes2.jpg'),'Parent',yesbut,'ButtonDownFcn','set(findobj('Tag','ConfirmReset'),'UserData',0); uiresume;', 'Tag','YesButton');
1606 axis(yesbut,'off');
1607 % Button No
1608 dimension = position;
1609 logospace = [77 95];
1610 dimension = [127, dimension(4)-logospace(2)-50, logospace];
1611 nobut = axes('Parent',confirmFig,'Units','pixels','Position',dimension,'Tag','No');
1612 image(imread('buttonno2.jpg'),'Parent',nobut,'ButtonDownFcn','set(findobj('Tag','ConfirmReset'),'UserData',1); uiresume;', 'Tag','NoButton');
1613 axis(nobut,'off');
1614
1615 uiwait;
1616
1617 if get(confirmFig,'UserData'), delete(confirmFig); return; end
1618 delete(confirmFig);
1619
1620 for i=1:length(selBlocks), set_param(selBlocks(i),'Description',''); set_param(selBlocks(i),'AttributesFormatString',''); end
1621 params = []; newparams = []; paramEnabled = []; oldparamEnabled = []; functionName = '';
1622 delete(findobj(gcf,'Parent',currPanel));
1623 paramcommand = RetrievePlist();
1624
1625 if findstr(paramcommand,'functionName='generic'')
1626     for i=1:length(selBlocks), set_param(selBlocks(i),'Name','generic'); end
1627 end
1628 eval(paramcommand)
1629 paramEnabled = zeros(1,nparams(newparams));
1630 for kk=1:length(selBlocks)
1631     set_param(selBlocks(kk),'Description',paramcommand);
1632 end
1633 AnnotationUpdate(newparams);
1634 buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
1635 buildplistFigureParam()
1636
1637 end
1638 %-----
1639
1640 %-----
1641 function KeepResultCallback(hObject,varargin)
1642 % Whenever the user clicks on the checkbox to maintain local results.
1643
1644 switch get(hObject,'Value')
1645 case 0
1646     % Cycle to restore proper color
1647     [meth,clas] = strtok(get_param(gcbh,'Tag'),' ');
1648     infoObj = eval([clas, '.getInfo('',getappdata(0,'ltpda_currFunctionName'),'')']);
1649     switch infoObj.mcategory
1650     case 'Constructor'
1651         set_param(gcbh,'BackgroundColor','green')
1652         set_param(gcbh,'MaskHelp','')
1653     case 'Output'
1654         set_param(gcbh,'BackgroundColor','lightBlue')
1655         set_param(gcbh,'MaskHelp','')
1656     otherwise
1657         set_param(gcbh,'BackgroundColor','red')
1658         set_param(gcbh,'MaskHelp','')
1659     end
1660 case 1
1661     set_param(gcbh,'BackgroundColor','magenta')
1662     set_param(gcbh,'MaskHelp','probe')
1663 end
1664
1665 end
1666 %-----
1667
1668 %-----
1669 function SetNameCallback(hObject,varargin)
1670 % Whenever the user clicks on the checkbox to maintain local results.

```

```

1670     setName = get(hObject,'Value');
1671     set(gcbh,'UserData',setName);
1672     set(gcbh,'UserDataPersistent','on');
1673
1674 end
1675 %-----
1676
1677 %-----
1678 function ApplyButtonCallback(hObject,varargin)
1679 % Whenever the user clicks on the 'Apply' button.
1680
1681
1682 % If it's executed automacally (upon button deletion), check whether
1683 % there's something to apply or not:
1684 if strcmp(get(hObject,'enable'),'off'), return; end
1685
1686 % This is to update the old params with those currently stored into
1687 % the block:
1688 temp = paramEnabled;
1689 paramcommand = get_param(gcbh,'Description');
1690 eval(paramcommand) % This overwrites also the current 'paramEnabled'
1691 oldparamEnabled = paramEnabled;
1692 paramEnabled = temp;
1693
1694 % Now to update the block with the latest changed newparams:
1695 paramcommand = BuildParamCommand(newparams);
1696 for kk=1:length(selBlocks)
1697     set_param(selBlocks(kk), 'Description', paramcommand);
1698 end
1699 AnnotationUpdate(newparams);
1700
1701 set(hObject,'enable','off')
1702
1703 % Redraw the panel only if the user hasn't changed selection or
1704 % clicked on a different tab:
1705 if nargin>2 && varargin{2}, buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData')); end
1706
1707 end
1708 %-----
1709
1710 %-----
1711 function UndoButtonCallback(varargin)
1712 % Whenever the user clicks on the 'Undo' button.
1713
1714     temp = paramEnabled;
1715     paramEnabled = oldparamEnabled;
1716     oldparamEnabled = temp;
1717
1718     paramcommand = BuildParamCommand(params);
1719     for kk=1:length(selBlocks)
1720         set_param(selBlocks(kk), 'Description', paramcommand);
1721     end
1722     AnnotationUpdate(params);
1723
1724 % This to swap params and newparams, so it's possible to undo the undo:
1725 temp = newparams;
1726 newparams = params;
1727 params = temp;
1728 clear temp;
1729
1730     buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
1731
1732 end
1733 %-----
1734
1735 %==== Input, Mux, Demux, From, Arithmetic block =====
1736 % =====
1737 %-----
1738
1739 function LoadInputParamCallback(varargin)
1740 % This callback is called whenever the user selects an Input block. The
1741 % callback draws the panel to set the only parameter associated to this
1742 % type of block: the ordinal number of AO.
1743
1744 if length(LTPDAinvar)<1
1745     % Text: 'Please load at least one AO in memory'
1746     uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDims(4)-200)/2 panelDims(4)-100 200
30] 'String','Please load at least one AO in memory','Visible','on','Style','text');
1747     % Button: 'Add data'
1748     uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[(panelDims(4)-200)/2 panelDims(4)-150 200 30] 'String','Add
data','FontSize',guiFontSize+1,'Visible','on','Enable','on','Callback',@AddButtonCallback,'Style','pushbutton');
1749 else
1750     % Text: 'Obj input:'
1751     uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','left','Position',[100 panelDims(4)-60 80 20] 'String','Object
input','Visible','on','Style','text');
1752     % Edit field:
1753     children = get_param(gcb,'Blocks');
1754     inputblock = strcat(gcb,'/',children{1});
1755     currAOinput = get_param(inputblock,'Value');
1756     uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[190 panelDims(4)-57 50
20] 'String',currAOinput,'Visible','on','Enable','on','Callback',@InputEditCallback,'Tag','inputEditField','Style','edit');

```

```

1757
1758 % Text: 'Objs list:'
1759 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','left','Position',[100 panelDims(4)-135 50 20],'String','Object
list:','Visible','on','Style','text');
1760 % Popup input objs list:
1761 xx = size(LTPDAinvar,1);
1762 listInput = cell(xx,1);
1763 for j=1:xx
1764     switch class(LTPDAinvar{j,1})
1765     case 'ao'
1766         anobject = LTPDAinvar{j,1};
1767         aoname = anobject.name;
1768         acreated = char(anobject.created);
1769         listInput{j,1} = [num2str(j) , ' . AO .' , aoname , '___' , acreated];
1770     case 'plist'
1771         paramNumb = nparams(LTPDAinvar{j,1});
1772         objcreated = char(LTPDAinvar{j,1}.created);
1773         listInput{j,1}=[num2str(j),' . PLIST .' , num2str(paramNumb), ' params','___',objcreated];
1774     case 'specwin'
1775         listInput{j,1}=[num2str(j), ' . SPECWIN .' , '___','no creation time'];
1776     case 'pzmodel'
1777         objcreated = char(LTPDAinvar{j,1}.created);
1778         listInput{j,1}=[num2str(j), ' . PZMODEL .' , '___',objcreated];
1779     otherwise
1780         if isa(LTPDAinvar{j,1},'ltpda_uo')
1781             objClass = class(LTPDAinvar{j,1});
1782             if ~isempty(LTPDAinvar{j,1}), listInput{j,1}=[num2str(j), ' . ',objClass]; end
1783         end
1784     end
1785 end
1786 for j=xx:-1:1
1787     if isempty(listInput{j,1})
1788         listInput(j,:)=[];
1789     end
1790 end
1791 inputPopup = uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[170 panelDims(4)-130 500
20],'String',listInput,'Visible','on','Enable','on','Callback',@InputPopupCallback,'Tag','inputPopupField','Style','popup');
1792
1793 % Set the popup list on the proper line:
1794 for ii = 1:size(listInput,1)
1795     if str2double(deblank(strtok(listInput{ii}))) == str2double(currAOinput), break; end
1796 end
1797 set(inputPopup,'Value',ii);
1798 end
1799 end
1800 %-----
1801 %-----
1802 %-----
1803 function InputEditCallback(hObject, varargin)
1804 % This callback is called whenever the user modify the input parameter
1805 % associated to an Input block
1806
1807 if ~isempty(varargin{1}), newValue = varargin{1};
1808 else newValue = str2double(get(hObject,'String'));
1809 end
1810 % Check it's an integer:
1811 newValue = round(newValue);
1812 % Check if the value is acceptable, ie:
1813 % - if it's <length(LTPDAinvar)
1814 if newValue>length(LTPDAinvar)
1815     newValue=length(LTPDAinvar);
1816 end
1817 % - if it's a LTPDA valid object:
1818 if ~isa(LTPDAinvar{newValue,1},'ltpda_uo')
1819     children = get_param(gcbb,'Blocks');
1820     inputblock = strcat(strcat(get(gcbb,'Path'),'/',get(gcbb,'Name')),'/',children{1});
1821     oldValue = get_param(inputblock,'Value');
1822     set(findobj('Tag','inputEditField'),'String',oldValue);
1823     return
1824 end
1825
1826 set(hObject,'String',num2str(newValue));
1827
1828 % Set the popup list on the proper line:
1829 inputPopup = findobj('Tag','inputPopupField');
1830 listInput = get(inputPopup,'String');
1831 for ii = 1:size(listInput,1)
1832     if str2double(deblank(strtok(listInput{ii}))) == newValue, break; end
1833 end
1834 set(inputPopup,'Value',ii);
1835
1836 % Object name for the annotation:
1837 objName = LTPDAinvar{newValue,1}.name;
1838 if numel(objName)>3 && strcmpi(objName,'none'), objName = class(LTPDAinvar{newValue,1}); end
1839 if numel(objName)>20, objName = ['...',LTPDAinvar{newValue,1}.name(end-20:end)]; end
1840 objName = [num2str(newValue), ': ',objName];
1841
1842 for i=1:length(selBlocks)
1843     % Since all input blocks are masks, and we want to change the
1844     % value deep inside a block INSIDE this mask, we can't use the
1845     % direct handles but the following:

```

```

1846     children{i}=get_param(selBlocks(i), 'Blocks');
1847     inputBlock=strcat(strcat(get(selBlocks(i), 'Path'), '/', get(selBlocks(i), 'Name')), '/', children{i}{1});
1848     set_param(inputBlock, 'Value', num2str(newValue));
1849     % Set the annotation:
1850     set_param(selBlocks(i), 'AttributesFormatString', objName);
1851 end
1852 %-----
1853 %-----
1854 %-----
1855 %-----
1856 function InputPopupCallback(hObject, varargin)
1857 % This callback is called whenever the user clicks on the input list in
1858 % the parameters panel associated to the 'Input Obj' block
1859
1860     val = get(hObject, 'Value');
1861     objLines = get(hObject, 'String');
1862     objnumb = str2double(deblank(strtok(objLines{val})));
1863
1864     InputEditCallback(findobj('Tag', 'inputEditField'), objnumb);
1865
1866 end
1867 %-----
1868 %-----
1869 %-----
1870 function AddButtonCallback(varargin)
1871 % Callback function: run when the user click the "Add data" button
1872
1873     filenames = uipickfiles('REFilter', '.txt');
1874     if ~isempty(filenames) && ~isnumeric(filenames(1,1))
1875         [x,y]=size(filenames);
1876         for j=1:y
1877             anobject={ao(filenames{j})};
1878             LTPDAinvar=[LTPDAinvar;anobject];
1879         end
1880         disp(sprintf(' + %g AOs added to the global variable LTPDAinvar', y))
1881         buildplistFigureParam([], [], get(findobj('Tag', 'paramsPage'), 'UserData'));
1882     end
1883 end
1884 %-----
1885 %-----
1886 %-----
1887 function LoadMuxParamCallback(varargin)
1888 % This callback is called whenever the user selects a Mux block (or
1889 % Demux). The callback opens the panel to set the only parameter
1890 % associated to this type of block: the number of inputs.
1891 % The function sets also the dimension of the block accordingly.
1892
1893     if strcmp(get(gcbh, 'Tag'), 'mux'),
1894         % Text: 'Number of inputs:'
1895         uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'HorizontalAlignment', 'left', 'Position', [100 panelDimens(4)-60 100 20], 'String', 'Number of
inputs:', 'Visible', 'on', 'Style', 'text');
1896         % Edit field:
1897         uicontrol('Parent', currPanel, 'Units', 'pixels', 'HorizontalAlignment', 'center', 'Position', [210 panelDimens(4)-57 50
20], 'String', get(gcbh, 'Inputs'), 'Visible', 'on', 'Enable', 'on', 'Callback', @MuxEditCallback, 'Style', 'edit');
1898     elseif strcmp(get(gcbh, 'Tag'), 'demux'),
1899         % Text: 'Number of outputs:'
1900         uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'HorizontalAlignment', 'left', 'Position', [100 panelDimens(4)-60 100 20], 'String', 'Number of
outputs:', 'Visible', 'on', 'Style', 'text');
1901         % Edit field:
1902         uicontrol('Parent', currPanel, 'Units', 'pixels', 'HorizontalAlignment', 'center', 'Position', [210 panelDimens(4)-57 50
20], 'String', get(gcbh, 'Outputs'), 'Visible', 'on', 'Enable', 'on', 'Callback', @MuxEditCallback, 'Style', 'edit');
1903     end
1904 end
1905 %-----
1906 %-----
1907 %-----
1908 function MuxEditCallback(hObject, varargin)
1909 % This callback is called whenever the user modify the number of inputs
1910 % associated to a Mux block (or outputs to a Demux)
1911
1912     newValue = str2double(get(hObject, 'String'));
1913     % Check it's an integer:
1914     newValue = round(newValue);
1915     set(hObject, 'String', num2str(newValue));
1916     for i=1:length(selBlocks)
1917         newDimens=get(selBlocks(i), 'Position');
1918         newDimens(4)=newDimens(2)+25*(newValue+1);
1919         if strcmp(get(gcbh, 'Tag'), 'mux')
1920             set(selBlocks(i), 'Inputs', num2str(newValue));
1921         elseif strcmp(get(gcbh, 'Tag'), 'demux')
1922             set(selBlocks(i), 'Outputs', num2str(newValue));
1923         end
1924     end
1925     set(selBlocks(i), 'Position', newDimens);
1926 end
1927 %-----
1928 %-----
1929 %-----
1930 function LoadFromBlockCallback(varargin)
1931 % This callback is called whenever the user selects a From block.
1932

```



```

1933 % Button: 'Find origin'
1934 uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-200)/2 panelDimens(4)-100 200 30],'String','Find
origin','FontSize',guiFontSize+1,'Visible','on','Enable','on','Callback',@FindOriginCallback,'Style','pushbutton');
1935 end
1936 %-----
1937
1938 %-----
1939 function FindOriginCallback(varargin)
1940 % This callback is called whenever click on the 'Find Origin' button,
1941 % having selected a 'From' block.
1942
1943 varName = get(gcbh,'GotoTag');
1944 origin = utils.prog.find_in_models(bdroot,'LookUnderMasks','all','BlockType','Goto','GotoTag',varName);
1945 if numel(origin)>1
1946 % Error text: there are multiple origins.
1947 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-400)/2 panelDimens(4)-160 400
30],'String','Error: there are multiple sources','Visible','on','Style','text');
1948 elseif numel(origin)==1
1949 % Text: full path of the origin.
1950 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-400)/2 panelDimens(4)-160 400
30],'String',origin{1},'Visible','on','Style','text');
1951 else
1952 % Error text: no origin found.
1953 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-400)/2 panelDimens(4)-160 400
30],'String','Error: no source found','Visible','on','Style','text');
1954 end
1955
1956
1957 end
1958 %-----
1959 %-----
1960
1961 %-----
1962 function arithmetic(varargin)
1963 % This callback is called whenever the user selects an arithmetic block.
1964
1965 delete(findobj(gcf,'Parent',currPanel))
1966 lineSpacing = getappdata(0,'ltpda_gui_LineSpacing');
1967 paramcommand = get_param(gcbh,'Description');
1968 noParamsReq=0;
1969 if isempty(paramcommand)
1970 paramcommand = RetrievePlist('arithmetic');
1971 selBlocks = utils.prog.gcbsh;
1972 for kk=1:length(selBlocks)
1973 set_param(selBlocks(kk),'Description',paramcommand);
1974 end
1975 end
1976 eval(paramcommand)
1977
1978 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[20 panelDimens(4)-55 100 20],'String','List of
input','HorizontalAlignment','left','FontSize',guiFontSize+1,'Visible','on','Style','text');
1979 y = nparams(newparams);
1980 for i=1:y
1981 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[7 panelDimens(4)-lineSpacing*(i+2)-4 8
20],'String','>','FontSize',guiFontSize+1,'FontWeight','Bold','Visible','on','Style','text');
1982 uicontrol('Parent',currPanel,'Units','pixels','Position',[20 panelDimens(4)-lineSpacing*(i+2) 100
20],'String',lower(newparams.params(i).key),'FontSize',guiFontSize+1,'Visible','on','Enable','on','UserData',i,'Callback',@editNameCallback,'Style','edit');
1983 % Remove input button
1984 if i>1
1985 uicontrol('Parent',currPanel,'Units','pixels','Position',[125 panelDimens(4)-lineSpacing*(i+2) 20 20],'String','-','Visible','on','Enable','on','Callback',
@RemParamCallback,'UserData',i,'Style','pushbutton');
1986 end
1987 end
1988 % Add input button
1989 uicontrol('Parent',currPanel,'Units','pixels','Position',[20 panelDimens(4)-40-lineSpacing*(y+2) 20 20],'String','+','Visible','on','Enable','on','Callback',
@AddParamCallback,'UserData','input','Style','pushbutton');
1990
1991 % Output name:
1992 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[20 panelDimens(4)-47-lineSpacing*(y+3) 140 20],'String','Output
name','HorizontalAlignment','left','FontSize',guiFontSize+1,'Visible','on','Style','text');
1993 outName = get(gcbh,'UserData');
1994 if isempty(outName), outName = get(gcbh,'Name'); end
1995 uicontrol('Parent',currPanel,'Units','pixels','Position',[20 panelDimens(4)-40-lineSpacing*(y+4) 100
20],'String',outName,'FontSize',guiFontSize+1,'Visible','on','Enable','on','UserData',i,'Callback','set(gcbh,''UserData'',get(gco,''String''));set(gcbh,''UserDataPersistent'', 'on''),'','Style',
);
1996
1997 % 'Type in the equation' text
1998 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[180 panelDimens(4)-50 400 20],'String','Type in the equation, using as an
example','HorizontalAlignment','left','FontSize',guiFontSize+1,'Visible','on','Style','text');
1999 % Example edit field
2000 uicontrol('Parent',currPanel,'TooltipString','This in an example of how to type in an equation','BackgroundColor',[0.97 0.97 0.97],'Units','pixels','Position',[180
panelDimens(4)-72 500
20],'String','(alpha+beta)/2+alpha^2','FontSize',guiFontSize+1,'Visible','on','Enable','on','Callback','set(gco,''String'', '(alpha+beta)/2+alpha^2'),'','Style','edit');
2001
2002 % 'output =' text
2003 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[180 panelDimens(4)-100 400 20],'String','Output
='','HorizontalAlignment','left','FontSize',guiFontSize+1,'Visible','on','Style','text');
2004 % Main edit field
2005 eqString = get_param(gcbh,'MaskDescription');
2006 if numel(eqString)>16 && strcmp(eqString(1:17),'Arithmetic block:'), eqString = ''; end
2007 uicontrol('Parent',currPanel,'BackgroundColor',[0.9 0.9 0.9],'Units','pixels','Position',[180 panelDimens(4)-300 500

```



```

2007, 'String', eqString, 'Max', 2, 'FontSize', guiFontSize+1, 'Visible', 'on', 'Enable', 'on', 'Callback', @eqEdit, 'Tag', 'equationField', 'Style', 'edit');
2008
2009     end
2010     %-----
2011     %-----
2012     %-----
2013     function eqEdit(hObject, varargin)
2014
2015         eq = get(hObject, 'String');
2016         if ~isempty(eq)
2017             if numel(eq)>40, eq(1:end-25) = []; eq = ['...', eq]; end
2018             for nn=1:length(selBlocks)
2019                 set_param(selBlocks(nn), 'MaskDescription', get(hObject, 'String'));
2020                 set_param(selBlocks(nn), 'AttributesFormatString', eq);
2021             end
2022         else
2023             eqString = get_param(gcbh, 'MaskDescription');
2024             if numel(eqString)>16 && strcmp(eqString(1:17), 'Arithmetic block:'), eqString = ''; end
2025             set(hObject, 'String', eqString);
2026         end
2027     end
2028     %-----
2029     %-----
2030
2031     %% ===== Errors =====
2032     % =====
2033     %-----
2034     function ErrorFuncNotSupported(varargin)
2035     % Recalled whenever the parameters call fails
2036
2037         % Text: '[...] Maybe it''s not supported yet?'
2038         uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'HorizontalAlignment', 'center', 'Position', [(panelDims(4)-300)/2 panelDims(4)-100-100 300
30], 'String', 'Error: unable to retrieve the parameters required by this function. Maybe it''s not supported yet?', 'Visible', 'on', 'Style', 'text');
2039     end
2040     %-----
2041     %-----
2042     %-----
2043     function ErrorSelectionInconsistent(varargin)
2044     % Recalled whenever the user selects inconsistent multiple blocks
2045         uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'HorizontalAlignment', 'center', 'Position', [(panelDims(4)-300)/2 panelDims(4)-100-100 300
30], 'String', 'Error: impossible to set unique parameters for the selected blocks', 'Visible', 'on', 'Style', 'text');
2046     end
2047     %-----
2048
2049     % =====
2050     %%
2051
2052 end

```

Home > m > gui > gltpda > pan4import.m

pan4import

PURPOSE ^

SYNOPSIS ^

function pan4import(varargin)

DESCRIPTION ^

```
=====
===== Panel to import data from the workspace =====
=====
$Id: pan4import.m,v 1.8 2008/08/30 12:27:08 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 ltpdagui =====

SUBFUNCTIONS ^

-  function UpdateListbox(varargin)
-  function hImportListCallback(hObject, varargin)
-  function ImportCallback(varargin)

SOURCE CODE ^

```
0001 function pan4import(varargin)
0002
0003 % =====
0004 % ===== Panel to import data from the workspace =====
0005 % =====
0006 % $Id: pan4import.m,v 1.8 2008/08/30 12:27:08 nicola Exp $
0007
0008
0009 global LTPDAinvar
0010
0011 ImportObjList='';
0012 UpdateListbox();
0013
0014 currPanel = varargin{1};
0015 panelDimens = get(currPanel, 'Position');
0016 backColor = get(currPanel, 'BackgroundColor');
0017
0018 DataPanPos = [20 panelDimens(4)-470 panelDimens(3)-40 450];
0019 hDataPanel = uipanel(...
0020     'Title','Data currently in the workspace',...
0021     'BackgroundColor',[1,1,1],...
0022     'FontSize',7,...
0023     'Units','pixels',...
0024     'Position',DataPanPos);
0025
0026 nbuttons = 3;
0027 buttonWidth = (DataPanPos(3)-10*(nbuttons+1))/nbuttons;
0028 buttonPos = [10 DataPanPos(4)-45 buttonWidth 25];
0029 hPlotButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Plot','FontSize',8,'Visible','on','Callback','', 'Style','pushbutton','enable','off');
0030 buttonPos(1) = buttonPos(1)+buttonWidth+8;
0031 hRefreshButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Refresh','FontSize',8,'Visible','on','Callback',@UpdateListbox, 'Style','pushbutton','enable','on');
0032 buttonPos(1) = buttonPos(1)+buttonWidth+8;
0033 hImportButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Import','FontSize',8,'Visible','on','Callback',@ImportCallback, 'Style','pushbutton','enable','on');
0034 buttonPos(1) = buttonPos(1)+buttonWidth+8;
0035 hExploreButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Explore','FontSize',8,'Visible','on','UserData',ImportObjList,'Callback',@hExploreButtonCallback, 'Style','pushbutton','enable','off');
0036 buttonPos(1) = buttonPos(1)+buttonWidth+8;
0037 hSaveOutButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Save','FontSize',8,'Visible','on','Enable','off','Callback',@SaveOutputCallback, 'Style','pushbutton');
0038 % clear nbuttons buttonWidth buttonPos
0039
0040
0041 textWidth = (DataPanPos(3)-10*(3))/2;
0042 textPos = [10 DataPanPos(4)-40 textWidth 25];
0043 uicontrol('Parent',hDataPanel,'BackgroundColor',get(hDataPanel,'BackgroundColor'),'HorizontalAlignment','center','Position',textPos,'String','Objects:', 'FontName','Times New Roman','FontSize',8,'FontWeight','normal','Visible','on','Style','text');
0044 clear textWidth textPos
0045
0046 nlists = 2;
0047 listWidth = (DataPanPos(3)-10*(nlists+1))/nlists;
0048 listPos = [10 DataPanPos(4)-315 listWidth 240];
0049 ImportObjList
0050 uicontrol('Parent',hDataPanel,'BackgroundColor',[0.95,0.95,0.95],'HorizontalAlignment','center','Position',listPos,'String',listInput,'Value',[], 'Max',10,'Min',0,'FontSize',8,'FontWeight','normal','Visible','on','Enable','on','Tag','ImportObjList','Callback',@hImportListCallback, 'Style',
0051
0050 listPos(1) = listPos(1)+listWidth+10;
0051 % Info panel:
0052 uicontrol('Parent',hDataPanel,'BackgroundColor',[0.94,0.94,1], 'Style','text','HorizontalAlignment', 'left','FontSize', 7,'Units','pixels','Position',listPos,'Tag','infoPanel');
0053 hImportListCallback(ImportObjList,1)
0054 clear DataPanPos nlists listWidth listPos
0055
0056
0057 %%%
0058 %-----
0059 function UpdateListbox(varargin)
0060 % Updates the listbox of objects in the current workspace
0061
0062 listInput = evalin('base','who');
0063
0064 for i=numel(listInput):-1:1
0065     obj = evalin('base',listInput{i});
0066     if ~isa(obj,'ltpda_uo') || isa(obj,'cell'), listInput(i)=[]; end
0067 end
0068
0069 if isempty(listInput)
0070     listInput = 'No LTPDA objects currently in the MATLAB workspace';
0071     set(findobj('Tag','ImportObjList'),'Enable','off','Value',[], 'String',listInput)
0072 else
0073     set(findobj('Tag','ImportObjList'),'Enable','on','Value',[], 'String',listInput)
0074 end
0075
0076 end
0077 %-----
```

```
0078
0079 %%
0080 %-----
0081 function hImportListCallback(hObject, varargin)
0082 % This callback is called whenever the user click upon one of the
0083 % objects in the workspace
0084
0085     if ~isempty(varargin{1}) && varargin{1}==1 && ~isa(listInput, 'char')
0086         % This is a call only to update the info panel
0087         val = get(hObject, 'Value');
0088         if length(val) == 1
0089             objLines = get(hObject, 'String');
0090             if isempty(objLines)
0091                 objtxt = '';
0092             else
0093                 objname = objLines{val};
0094                 obj = evalin('base', objname);
0095                 objtxt = display(obj);
0096             end
0097         else
0098             objtxt = '';
0099         end
0100         set(findobj('Tag', 'infoPanel'), 'String', objtxt)
0101     elseif ~isa(listInput, 'char')
0102         strcmp(get(gcf, 'SelectionType'), 'normal')
0103         val = get(hObject, 'Value');
0104         if length(val) == 1
0105             objLines = get(hObject, 'String');
0106             objname = objLines{val};
0107             obj = evalin('base', objname);
0108             objtxt = display(obj);
0109         else
0110             objtxt = '';
0111         end
0112         set(findobj('Tag', 'infoPanel'), 'String', objtxt)
0113     end
0114     if strcmp(get(gcf, 'SelectionType'), 'open') && ~isempty(get(hObject, 'String'))
0115         val = get(hObject, 'Value');
0116         objLines = get(hObject, 'String');
0117         objname = objLines{val};
0118         obj = evalin('base', objname);
0119         try %ok<ALIGN>
0120             figure
0121             plot(obj);
0122         catch
0123             end
0124         end
0125     end
0126 end
0127
0128 %-----
0129
0130 %-----
0131
0132 function ImportCallback(varargin)
0133 % This callback is called whenever the user click the 'Import' button
0134
0135     objList = findobj('Tag', 'ImportObjList');
0136     val = get(objList, 'Value');
0137     objLines = get(objList, 'String');
0138     for i=1:numel(val)
0139         objname = objLines{val(i)};
0140         obj = evalin('base', objname);
0141         nobj = numel(obj);
0142         if nobj>1
0143             for jj=1:nobj
0144                 objj = {obj{ij}, 0};
0145                 LTPDAinvar = [LTPDAinvar; objjj];
0146             end
0147         else
0148             obj = {obj , 0};
0149             LTPDAinvar = [LTPDAinvar; objj];
0150         end
0151     end
0152     set(findobj('Tag', 'ImportObjList'), 'Value', [])
0153 end
0154
0155 %-----
0156
0157 %%
0158 end
```

Home > m > gui > gltpda > pan5output.m

pan5output

PURPOSE ^

=====

SYNOPSIS ^

function pan5output(varargin)

DESCRIPTION ^

=====
Panel for the outputs
=====
This is the GUI m-file to show that the panel relative to the outputs
\$Id: pan5output.m,v 1.16 2008/08/12 18:50:06 nicola Exp \$
=====

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpdagui =====

SUBFUNCTIONS ^

- function UpdateListbox(varargin)
- function hResultListCallback(hObject, varargin)
- function hClearButtonCallback(varargin)
- function hExportButtonCallback(varargin)
- function hExploreButtonCallback(varargin)

SOURCE CODE ^

```
0001 function pan5output(varargin)
0002 % =====
0003 % ===== Panel for the outputs =====
0004 % =====
0005 % =====
0006 % This is the GUI m-file to show that the panel relative to the outputs
0007 %
0008 % $Id: pan5output.m,v 1.16 2008/08/12 18:50:06 nicola Exp $
0009
0010 global LTPDAoutvar
0011
0012 InputObjList=''; OutputObjList='';
0013 UpdateListbox();
0014
0015 currPanel = varargin{1};
0016 panelDimens = get(currPanel, 'Position');
0017 backColor = get(currPanel, 'BackgroundColor');
0018
0019 DataPanPos = [20 panelDimens(4)-320 panelDimens(3)-40 285];
0020 hDataPanel = uipanel(...
0021     'Title','Data Control Panel',...
0022     'BackgroundColor',[1,1,1],...
0023     'FontSize',7,...
0024     'Units','pixels',...
0025     'Position',DataPanPos);
0026
0027 nbuttons = 5;
0028 buttonWidth = (DataPanPos(3)-10*(nbuttons+1))/nbuttons;
0029 buttonPos(4) = [10 DataPanPos(4)-45 buttonWidth 25];
0030 hExportButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Export to workspace','FontSize',8,'Visible','on','Callback',@hExportButtonCallback,'Style','pushbutton','enable','on');
0031 buttonPos(1) = buttonPos(1)+buttonWidth+10;
0032 hRemoveButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Remove','FontSize',8,'Visible','on','Callback',@hRemoveButtonCallback,'Style','pushbutton','enable','off');
0033 buttonPos(1) = buttonPos(1)+buttonWidth+10;
0034 hRepoButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Send to repository','FontSize',8,'Visible','on','Callback',@hClearButtonCallback,'Style','pushbutton','enable','off');
0035 buttonPos(1) = buttonPos(1)+buttonWidth+10;
0036 hExploreButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Explore','FontSize',8,'Visible','on','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton','enable','on');
0037 hSaveOutButton = uicontrol('Parent',hDataPanel,'HorizontalAlignment','center','Position',buttonPos,'String','Save workspace','FontSize',8,'Visible','on','Enable','on','Callback',@hSaveWorkspaceCallback,'Style','pushbutton');
0038 buttonPos(1) = buttonPos(1)+buttonWidth+10;
0039 hClearButton = uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Clear output','FontSize',8,'Visible','on','Enable','on','Callback',@hClearButtonCallback,'Style','pushbutton');
0040 clear nbuttons buttonWidth buttonPos
0041
0042
0043 ntexts = 2;
0044 textWidth = (DataPanPos(3)-10*(ntexts+1))/ntexts;
0045 textPos = [10 DataPanPos(4)-80 textWidth 25];
0046 % hListAOText = uicontrol('Parent',hDataPanel,'BackgroundColor',get(hDataPanel,'BackgroundColor'),'HorizontalAlignment','center','Position',textPos,'String','Input:','FontName','Times New Roman','FontSize',8,'FontWeight','normal','Visible','on','Style','text');
0047 textPos(1) = textPos(1)+textWidth+10;
0048 hResultAOText = uicontrol('Parent',hDataPanel,'BackgroundColor',get(hDataPanel,'BackgroundColor'),'HorizontalAlignment','center','Position',textPos,'String','Output:','FontName','Times New Roman','FontSize',8,'FontWeight','normal','Visible','on','Style','text');
0049 clear ntexts textWidth textPos
0050
0051
0052 nlists = 2;
0053 listWidth = (DataPanPos(3)-10*(nlists+1))/nlists;
0054 listPos = [10 DataPanPos(4)-275 listWidth 280];
0055 % info panel
0056 uicontrol('Parent',hDataPanel,'BackgroundColor',[0.94,0.94,1],'Style','listbox','HorizontalAlignment','left','FontSize',8,'Units','pixels','Position',listPos,'Tag','infoPanel');
0057 listPos(1) = listPos(1)+listWidth+10;
0058 OutputObjList =
0059 uicontrol('Parent',hDataPanel,'BackgroundColor',[0.95,0.95,0.95],'HorizontalAlignment','center','Position',listPos,'String',ListResults,'Value',[],'Max',10,'Min',1,'FontSize',8,'FontWeight','normal','Visible','on','Callback',@hResultListCallback,'Tag','OutputObjList','Style',
0059 hResultListCallback(OutputObjList,1)
0060 clear DataPanPos nlists listWidth listPos
0061
0062
0063 %%
0064 %-----
0065 function UpdateListbox(varargin)
0066 % Updates the LTPDAoutvar listbox
```

```
0067
0068
0069 % Update the list of results AOs
0070 xx = size(LTPDAoutvar,1);
0071 listResults = cell(xx,1);
0072 if xx>0
0073     for j=1:xx
0074         listResults{j,1} = num2str(j);
0075         [rows,columns] = size(LTPDAoutvar{j,1});
0076         if (rows>1 && columns==1) || (rows==1 && columns>1), listResults{j,1} = [listResults{j,1}, ' (Vector ',num2str(rows),'x',num2str(columns),' ) '];
0077         elseif rows>1 && columns>1, listResults{j,1} = [listResults{j,1}, ' (Matrix ',num2str(rows),'x',num2str(columns),' ) '];
0078         end
0079         switch class(LTPDAoutvar{j,1})
0080             case 'ao'
0081                 anobject = LTPDAoutvar{j,1};
0082                 aoname = anobject.name;
0083                 aocreated = char(anobject.created);
0084                 listResults{j,1} = [listResults{j,1}, ' . AO .', aoname, ' __', aocreated];
0085             case 'plist'
0086                 paramNumb = nparams(LTPDAoutvar{j,1});
0087                 objcreated = get(LTPDAoutvar{j,1}.created,'time_str');
0088                 listResults{j,1}=[listResults{j,1}, ' . PLIST .',num2str(paramNumb),' params','__',objcreated];
0089             case 'specwin'
0090                 listResults{j,1}=[listResults{j,1}, ' . SPECWIN .','__','no creation time'];
0091             case 'pzmodel'
0092                 objcreated = get(LTPDAoutvar{j,1}.created,'time_str');
0093                 listResults{j,1}=[listResults{j,1}, ' . PZMODEL .','__',objcreated];
0094             case 'mir'
0095                 objcreated = get(LTPDAoutvar{j,1}.created,'time_str');
0096                 listResults{j,1}=[listResults{j,1}, ' . MIR .','__',objcreated];
0097             otherwise
0098                 if isa(LTPDAoutvar{j,1},'ltpda_uo')
0099                     objclass = class(LTPDAoutvar{j,1});
0100                     if ~isempty(LTPDAoutvar{j,1}), listResults{j,1} = [listResults{j,1}, ' . ',objclass]; end
0101                 end
0102             end
0103         end
0104         for j=xx:-1:1
0105             if isempty(listResults{j,1})
0106                 listResults(j,:)=[];
0107             end
0108         end
0109     else
0110         listResults{1,1}='The memory is empty';
0111     end
0112     OutputObjList = findobj('Tag','OutputObjList');
0113     if ~isempty(OutputObjList)
0114         set(OutputObjList,'Value',1)
0115         set(OutputObjList,'String',listResults)
0116     end
0117 end
0118
0119 %=====
0120 end
0121
0122 %-----
0123
0124 %%
0125 %-----
0126 function hResultListCallback(hObject, varargin)
0127 % This callback is called whenever the user double click upon one of
0128 % the AOs in use.
0129 if ~isempty(varargin{1}) && varargin{1}==1 && size(LTPDAoutvar,1)>0
0130     val = get(hObject,'Value');
0131     if length(val) == 1
0132         objLines = get(hObject, 'String');
0133         if isempty(objLines)
0134             objtxt = '';
0135         else
0136             objnumb = str2double(deblank(strtok(objLines{val})));
0137             objtxt = display(LTPDAoutvar{objnumb});
0138         end
0139     else
0140         objtxt = '';
0141     end
0142     set(findobj('Tag','infoPanel'),'String',objtxt)
0143 elseif size(LTPDAoutvar,1)>0
0144     if strcmp(get(gcf,'SelectionType'),'normal')
0145         val = get(hObject,'Value');
0146         if length(val) == 1
0147             objLines = get(hObject, 'String');
0148             objnumb = str2double(deblank(strtok(objLines{val})));
0149             objtxt = display(LTPDAoutvar{objnumb});
0150         else
0151             objtxt = '';
0152         end
0153         set(findobj('Tag','infoPanel'),'String',objtxt)
0154     end
0155     if strcmp(get(gcf,'SelectionType'),'open') && ~isempty(get(hObject, 'String'))
0156         try %ok<ALIGN>
0157             currIndex = get(hObject,'Value');
0158             figure
0159             plot(LTPDAoutvar{currIndex,1});
0160         catch
0161             close(gcf)
0162         end
0163     end
0164 end
0165
0166 end
0167
0168 %-----
0169 %
0170 % function SaveWorkspaceCallback(varargin)
0171 % Callback function: run when the user click the "Clear all data"
0172 % Button
0173 %
0174 % try
0175 %     [FileName,PathName,FilterIndex] = uiputfile('*.mat','Save Workspace As');
0176 %     if ~isa(FileName,'double'), save(strcat(PathName,FileName),'LTPDAoutvar'); disp('workspace saved'); end
0177 %     disp('Output saved')
0178 % catch
0179 %     end
0180 % end
0181 %
0182 %-----
0183 %
0184 function hClearButtonCallback(varargin)
0185 % Callback function: run when the user click the "Clear all data"
0186 % Button
0187 %
0188 % LTPDAoutvar={};
0189 % set(OutputObjList,'Value',1)
0190 % set(findobj('Tag','infoPanel'),'String','')
0191 % UpdateListBox();
0192 end
0193
0194 %-----
0195
0196 %-----
```

```
0197 function hExportButtonCallback(varargin)
0198 % Callback function: run when the user click the "Remove data" button
0199
0200     currSelObjs = get(OutputObjList, 'Value');
0201     listObjs = get(OutputObjList, 'String');
0202     if ~strcmp(listObjs, 'The memory is empty')
0203         selectedObjs = {};
0204         for i=1:numel(currSelObjs)
0205             objSelect = str2double(strtok(listObjs{currSelObjs(i),1}, '.'));
0206             selectedObjs = [selectedObjs, LTPDAoutvar(objSelect,1)];
0207         end
0208         if ~isempty(selectedObjs)
0209             for i=1:numel(selectedObjs)
0210                 objName = [class(selectedObjs{i}), ' ', selectedObjs{i}.name];
0211                 objName = genvarname(objName, evalin('base', 'who'));
0212                 assignin('base', objName, selectedObjs{i});
0213                 disp(['Saved to workspace with name ', objName])
0214             end
0215         end
0216     end
0217 end
0218 %-----
0219
0220 %-----
0221
0222 function hExploreButtonCallback(varargin)
0223 % Callback function: run when the user click the "Update list" button
0224
0225     currSelObjs = get(OutputObjList, 'Value');
0226     listObjs = get(OutputObjList, 'String');
0227     if ~strcmp(listObjs, 'The memory is empty')
0228         selectedObjs = {};
0229         for i=1:numel(currSelObjs)
0230             objSelect = str2double(strtok(listObjs{currSelObjs(i),1}, '.'));
0231             selectedObjs = [selectedObjs, LTPDAoutvar(objSelect,1)];
0232         end
0233         if ~isempty(selectedObjs)
0234             ltpda_explorer(selectedObjs);
0235         else
0236             ltpda_explorer(LTPDAoutvar);
0237         end
0238     end
0239     UpdateListbox();
0240 end
0241
0242 %-----
0243
0244 %-----
0245
0246 %%
0247 %%
0248 end
```

pan6globals

PURPOSE ^

=====

SYNOPSIS ^

function pan6globals(varargin)

DESCRIPTION ^

```
=====
===== Panel for the globals =====
=====
This is the GUI m-file to show that the panel relative to the global
variables set by the user.

$Id: pan6globals.m,v 1.6 2008/08/30 12:32:25 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdagui](#) =====

SUBFUNCTIONS ^

-  [function](#) buildGlobalPanel(varargin)
-  [function](#) ContinuousParamCheck(varargin)
-  [function](#) AddGlobal(varargin)
-  [function](#) RemoveGlobal(hObject,varargin)
-  [function](#) globalName(hObject,varargin)
-  [function](#) globalValue(hObject,varargin)
-  [function](#) updateGlobalBlock(varargin)

SOURCE CODE ^

```
0001 function pan6globals(varargin)
0002
0003 % =====
0004 % ===== Panel for the globals =====
0005 % =====
0006 % This is the GUI m-file to show that the panel relative to the global
0007 % variables set by the user.
0008 %
0009 % $Id: pan6globals.m,v 1.6 2008/08/30 12:32:25 nicola Exp $
0010
0011 global gl currSys
0012 currPanel = varargin{1};
0013 panelDimens = get(currPanel, 'Position');
0014 backColor = get(currPanel, 'BackgroundColor');
0015 selBlocks = 0;
0016 enlargedFont = 0;
0017 currSys = '';
0018
0019 alltimers = timerfindall;
0020 stop(alltimers(1));
0021 set(alltimers(1), 'TimerFcn', @ContinuousParamCheck, 'Period', 0.3);
0022 start(alltimers(1));
0023
0024 %-----
0025 function buildGlobalPanel(varargin)
0026 %-----
0027
0028 globBlock = find_system(bdroot, 'SearchDepth', 1, 'BlockType', 'SubSystem', 'Tag', 'globals');
0029 if isempty(globBlock) || (~isempty(varargin) && isnumeric(varargin{3}) && varargin{3}== -1)
0030     gl = struct();
0031     updateGlobalBlock()
0032 else
0033     gl = get_param(globBlock{1}, 'UserData');
0034 end
0035
0036 % if isempty(gl) || (~isempty(varargin) && isnumeric(varargin{3}) && varargin{3}== -1), gl=struct(); end
0037
0038 panelDimens = get(currPanel, 'Position');
0039 delete(findobj(gcf, 'Parent', currPanel))
0040 varnames = fieldnames(gl);
0041 y = numel(varnames);
0042
0043 % 'List of global variables' text:
0044 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'Position', [20 panelDimens(4)-40 panelDimens(3)-
40 20], 'String', 'List of global
variables:', 'HorizontalAlignment', 'center', 'FontSize', enlargedFont+10, 'FontWeight', 'bold', 'Visible', 'on', 'Style', 'text');
0045 % 'Global names' text:
0046 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'Position', [panelDimens(3)/2-150-20
panelDimens(4)-70 150
20], 'String', 'Names:', 'HorizontalAlignment', 'right', 'FontSize', enlargedFont+9, 'Visible', 'on', 'Style', 'text');
0047 % 'Global values' text:
0048 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'Units', 'pixels', 'Position', [panelDimens(3)/2+20 panelDimens(4)-70
150 20], 'String', 'Values:', 'HorizontalAlignment', 'left', 'FontSize', enlargedFont+9, 'Visible', 'on', 'Style', 'text');
```

```

0049
0050 for i=1:y
0051 % % 'gl.' text:
0052 % uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[panelDims(3)/2-200-20-15
panelDims(4)-20-24*(i+2)-5 12 20],'String','gl.','FontSize',enlargedFont+8,'Visible','on','Enable','off','Style','text');
0053 % Variable name edit field:
0054 uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDims(3)/2-200-20 panelDims(4)-20-24*(i+2) 200
20],'String',[ 'gl.',varnames{i}],'FontSize',enlargedFont+8,'Visible','on','Enable','on','UserData',i,'Callback',@globalName,'Style',
);
0055 % Variable value edit field:
0056 currVal = gl.(varnames{i});
0057 if iscell(currVal), currVal = cell2str(currVal);
0058 else currVal = mat2str(currVal);
0059 end
0060 uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDims(3)/2+20 panelDims(4)-20-24*(i+2) 200
20],'String',currVal,'FontSize',enlargedFont+8,'Visible','on','Enable','on','UserData',i,'Callback',@globalValue,'Style','edit');
0061 % Remove input button
0062 uicontrol('Parent',currPanel,'Units','pixels','Position',[700 panelDims(4)-20-24*(i+2) 20 20],'String','-
','Visible','on','Enable','on','Callback',@RemoveGlobal,'UserData',i,'Style','pushbutton');
0063 end
0064
0065 % Add input button
0066 uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDims(3)/2-10 panelDims(4)-50-24*(y+2) 20
20],'String','+', 'TooltipString','Add a global variable','Visible','on','Enable','on','Callback',
@AddGlobal,'Style','pushbutton');
0067
0068 % Clear all button:
0069 uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[panelDims(3)-100 10 80
16],'FontAngle','italic','FontSize',enlargedFont+8,'String','Clear all','TooltipString','Clear all the global
variables','Visible','on','Callback',{@buildGlobalPanel,-1},'Style','pushbutton');
0070
0071 end
0072 %-----
0073 %
0074 %
0075 %%
0076 %-----
0077 function ContinuousParamCheck(varargin)
0078 % This is the function to execute a continuous check on the status
0079 % of the current selection (system and block).
0080
0081 % Clear if nothing's selected
0082 if isempty(bdroot) || strcmp(get_param(bdroot,'BlockDiagramType'),'library') ||
isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
0083 delete(findobj(gcf,'Parent',currPanel))
0084 currSys = gcs;
0085 gl = struct();
0086 return
0087 else
0088 if ~strcmp(gcs,currSys)
0089 % The selection changed.
0090 currSys = gcs;
0091 buildGlobalPanel();
0092 end
0093 end
0094
0095 if ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
0096 % It's selected a ltpda simulink analysis diagram.
0097 if ~strcmp(gcs,currSys)
0098 % The selection changed.
0099 currSys = gcs;
0100 buildGlobalPanel();
0101 end
0102 end
0103 end
0104 %-----
0105 %
0106 %
0107 %%
0108 %-----
0109 function AddGlobal(varargin)
0110
0111 newGlob = 'global';
0112 ii = 1;
0113 while isfield(gl,[newGlob,num2str(ii)]), ii = ii+1; end
0114 newGlob = [newGlob,num2str(ii)];
0115 gl.(newGlob) = [];
0116 updateGlobalBlock()
0117 buildGlobalPanel()
0118
0119 end
0120 %-----
0121 %
0122 %-----
0123 function RemoveGlobal(hObject,varargin)
0124
0125 currGlob = get(hObject,'UserData');
0126 varnames = fieldnames(gl);
0127 gl = rmfield(gl,varnames{currGlob});
0128 updateGlobalBlock()
0129 buildGlobalPanel()
0130
0131 end
0132 %-----
0133 %
0134 %-----
0135 function globalName(hObject,varargin)
0136
0137 currGlob = get(hObject,'UserData');
0138 varnames = fieldnames(gl);
0139 newGlobName = get(hObject,'String');
0140 if numel(newGlobName)<4 || ~strcmp(newGlobName(1:3),'gl.'), newGlobName = ['gl.',strrep(newGlobName, '.', '')]; end
0141
0142 newGlobName = ['gl.',genvarname(newGlobName(4:end))];
0143 if isempty(get(hObject,'String')) || isfield(gl,newGlobName(4:end)),
set(hObject,'String',[ 'gl.',varnames{currGlob}]); return; end
0144
0145 set(hObject,'String',newGlobName);
0146 gl = utils.prog.rmfield(gl,varnames{currGlob},newGlobName(4:end));
0147 updateGlobalBlock()
0148 buildGlobalPanel()
0149
0150 end
0151 %-----
0152 %
0153 %-----
0154 function globalValue(hObject,varargin)
0155
0156 currGlob = get(hObject,'UserData');

```



```

0157     newValue = get(hObject,'String');
0158     varnames = fieldnames(gl);
0159     try newValue = eval(newValue); catch end
0160     gl.(varnames{currGlob}) = newValue;
0161     updateGlobalBlock()
0162     buildGlobalPanel()
0163
0164 end
0165 %-----
0166
0167 %-----
0168 function updateGlobalBlock(varargin)
0169
0170     varNames = fieldnames(gl);
0171
0172     if numel(varNames)==0
0173         % Remove the block if no globals are left:
0174         globBlock = find_system(bdroot,'BlockType','SubSystem','Tag','globals');
0175         if ~isempty(globBlock)
0176             delete_block(globBlock)
0177             return;
0178         end
0179     else
0180         globBlock = find_system(bdroot,'SearchDepth',1,'BlockType','SubSystem','Tag','globals');
0181         try globBlock = globBlock{1}; catch end
0182         % Check if the globals block exists:
0183         if isempty(globBlock)
0184             globBlock = add_block('ltpda_library/Commonly Used Blocks/Globals',[bdroot '/Globals']);
0185             annotation = find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model');
0186             position = get_param(annotation,'Position');
0187             set_param(globBlock,'Position',[position(1)+20 position(2)-25 position(1)+150 position(2)-10 ]);
0188             set_param(globBlock,'DropShadow','on')
0189         end
0190
0191         % Update the globals block:
0192         set_param(globBlock,'UserData',gl)
0193         set_param(globBlock,'UserDataPersistent','on')
0194     end
0195
0196
0197 end
0198 %-----
0199
0200 %%
0201 end

```

pan7partial

PURPOSE ^

=====

SYNOPSIS ^

function pan7partial(varargin)

DESCRIPTION ^

```
=====
===== Panel for partial execution =====
=====
$Id: pan7partial.m,v 1.3 2008/08/30 12:27:08 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdagui](#) =====

This function is called by:

 [ltpdagui](#) =====

SUBFUNCTIONS ^

-  [function ContinuousCheck\(varargin\)](#)
-  [function UpdatePanel\(varargin\)](#)
-  [function Run1\(varargin\)](#)
-  [function Run2\(varargin\)](#)
-  [function Run3\(varargin\)](#)

SOURCE CODE ^

```
0001 function pan7partial(varargin)
0002
0003 % =====
0004 % ===== Panel for partial execution =====
0005 % =====
0006 %
0007 % $Id: pan7partial.m,v 1.3 2008/08/30 12:27:08 nicola Exp $
0008
0009 scaleX = getappdata(0, 'ltpda_gui_scalefactor');
0010 guiFontSize = round(8*scaleX);
0011
0012 alltimers = timerfindall;
0013 stop(alltimers(2));
0014 set(alltimers(2), 'TimerFcn', @ContinuousCheck, 'Period', 1);
0015 start(alltimers(2));
0016
0017 %%
0018
0019 currPanel = varargin{1};
0020 panelDimens = get(currPanel, 'Position');
0021 backColor = get(currPanel, 'BackgroundColor');
0022
0023 % Current system:
0024
0025 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Units', 'normalized', 'Po
0026 , [0.17 0.89 0.125 0.040], 'String', 'Current system:', 'FontName', 'Times New
0027 Roman', 'FontSize', guiFontSize+4, 'FontWeight', 'normal', 'Visible', 'on', 'Style', 'text');
0028
0029 currsystem = bdroot;
0030 if isempty(currsystem) ||
0031 isempty(find_system(currsystem, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda model')), currsystem =
0032 'None'; end
0033
0034 uicontrol('Parent', currPanel, 'BackgroundColor', backColor, 'HorizontalAlignment', 'left', 'Units', 'normalized', 'Po
0035 , [0.3 0.89 0.1875 0.040], 'String', currsystem, 'FontName', 'Times New
0036 Roman', 'FontSize', guiFontSize+4, 'FontWeight', 'bold', 'Visible', 'on', 'UserData', 'currsystem', 'Style', 'text');
```

```

0028
0029 % Current block:
0030
0031 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Units','normalized','Po
,[0.5 0.89 0.125 0.040],'String','Current block:','FontName','Times New
Roman','FontSize',guiFontSize+4,'FontWeight','normal','Visible','on','Style','text');
0031
0032 uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Units','normalized','Po
,[0.62 0.89 0.1875 0.040],'String','','FontName','Times New
Roman','FontSize',guiFontSize+4,'FontWeight','bold','Visible','on','UserData','currblock','Style','text');
0032
0033
0034 % =====
0035 FirstPanPos = [.0125 .56 .975 .26];
0036 hFirstPanel = uipanel('Parent',currPanel,'BackgroundColor',backColor,'FontSize',guiFontSize-
1,'Units','normalized','Position',FirstPanPos);
0037 logospace = [53 55]*scaleX;
0038 icon1 = axes('Parent',hFirstPanel,'Units','normalized','Position',[.03 .45 0.3 0.3]);
0039 iconSize = getpixelposition(icon1); panelSize = getpixelposition(hFirstPanel); iconSize(3:4) =
logospace; iconSize(2) = (panelSize(4)-iconSize(4))/2; setpixelposition(icon1,iconSize);
0040 image(imread('iconpar1.jpg'),'Parent',icon1,'Tag','partial1');
0041 axis(icon1,'off');
0042 clear logospace iconSize panelSize
0043
0044 firstText = sprintf('Execution from the beginning to the current block.\n\nThis option is always
possible: are executed only those blocks parent of the current one, and the execution is stopped as soon
as the current one is reached.');
```

```

0045
0046 uicontrol('Parent',hFirstPanel,'BackgroundColor',backColor,'Units','normalized','HorizontalAlignment','left','
,[0.15 0.2 0.8
0.6],'String',firstText,'FontSize',guiFontSize,'Visible','on','Style','text','Tag','infotext1');
```

```

0047 % =====
0048 SecondPanPos = [.0125 .29 .975 .26];
0049 hSecondPanel = uipanel('Parent',currPanel,'BackgroundColor',backColor,'FontSize',guiFontSize-
1,'Units','normalized','Position',SecondPanPos);
0050 logospace = [53 55]*scaleX;
0051 icon2 = axes('Parent',hSecondPanel,'Units','normalized','Position',[.03 .45 0.3 0.3]);
0052 iconSize = getpixelposition(icon2); panelSize = getpixelposition(hFirstPanel); iconSize(3:4) =
logospace; iconSize(2) = (panelSize(4)-iconSize(4))/2; setpixelposition(icon2,iconSize);
0053 image(imread('iconpar2.jpg'),'Parent',icon2,'Tag','partial2');
```

```

0054 axis(icon2,'off');
```

```

0055 clear logospace iconSize panelSize
0056
0057 secondText = sprintf('Execution of the current block only.\n\nThis option is possible only if all
the inputs necessary to the currently selected block are available; this means that the model must have
already been executed and either all intermediate results are maintained, or just those necessary to the
current block.');
```

```

0058
0059 uicontrol('Parent',hSecondPanel,'BackgroundColor',backColor,'Units','normalized','HorizontalAlignment','left',
,[0.15 0.2 0.8
0.6],'String',secondText,'FontSize',guiFontSize,'Visible','on','Style','text','Tag','infotext2');
```

```

0060 % =====
0061 ThirdPanPos = [.0125 .02 .975 .26];
0062 hThirdPanel = uipanel('Parent',currPanel,'BackgroundColor',backColor,'FontSize',guiFontSize-
1,'Units','normalized','Position',ThirdPanPos);
0063 logospace = [53 55]*scaleX;
0064 icon3 = axes('Parent',hThirdPanel,'Units','normalized','Position',[.03 .45 0.3 0.3]);
0065 iconSize = getpixelposition(icon3); panelSize = getpixelposition(hFirstPanel); iconSize(3:4) =
logospace; iconSize(2) = (panelSize(4)-iconSize(4))/2; setpixelposition(icon3,iconSize);
0066 image(imread('iconpar3.jpg'),'Parent',icon3,'Tag','partial3');
```

```

0067 axis(icon3,'off');
```

```

0068 clear logospace iconSize panelSize
0069
0070 thirdText = sprintf('Execution from the current block to the end.\n\nThis option is possible only
if all the inputs necessary to the currently selected block are available; this means that the model must
have already been executed and either all intermediate results are maintained, or just those necessary to
the current block.\nAfter the current block the execution continues, updating all the following blocks');
```

```

0071
0072 uicontrol('Parent',hThirdPanel,'BackgroundColor',backColor,'Units','normalized','HorizontalAlignment','left','
,[0.15 0.2 0.8
0.6],'String',thirdText,'FontSize',guiFontSize,'Visible','on','Style','text','Tag','infotext3');
```

```

0073
0074 UpdatePanel(1);
0075
0076
0077 %%
0078 %-----
0079
0080 function ContinuousCheck(varargin)
0081 % This is the function to execute a continuous timer check
0082
0083 % Keep alive the first timer:
0084 alltimers = timerfindall;
0085 if strcmp(get(alltimers(1),'Running'),'off')
0086     start(alltimers(1))
0087 end
0088
0089 currsys = bdroot;
0090 oldsys = get(findobj('UserData','currsystem'),'String');
0091 currselection = utils.prog.gcbsh;
0092 if numel(currselection)>1, UpdatePanel(1); return; end
0093 if isempty(currsys) || isempty(currselection), UpdatePanel(1); return; end
0094 currblock = get_param(currselection,'Name');
0095 oldblock = get(findobj('UserData','currblock'),'String');
0096 if strcmp(currsys,oldsys) && strcmp(currblock,oldblock), return; end
0097
```

```

0098     if ~isempty(currsys) &&
~isempty(find_system(currsys, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda model'))
0099     currchild = find_system(currselection, 'LookUnderMasks', 'all', 'BlockType', 'M-S-Function');
0100     if isempty(currchild) || ~strcmp(get_param(currchild, 'FunctionName'), 'ltpdasim')
0101         % disp('*** The model is valid, but the selected block is not an LTPDA function block.')
0102         UpdatePanel(2)
0103         return;
0104     else
0105         annotation = find_system(bdroot, 'FindAll', 'on', 'Type', 'Annotation', 'Tag', 'ltpda
model');
0106         execHistory = get_param(annotation, 'UserData');
0107         if isempty(execHistory)
0108             % disp('*** The model is valid, but has never been run.')
0109             UpdatePanel(3)
0110             return;
0111         else
0112             blockIndex = -1;
0113             for i=1:size(execHistory,1), if execHistory{i,2}==currselection, blockIndex=i;
break; end; end
0114             if blockIndex==-1 % the current block is not included in the execution history
0115                 % disp('*** The model is valid, but the selected block is not included in the
execution history.')
0116                 UpdatePanel(3)
0117                 return;
0118             end
0119             for i=3:size(execHistory,2)
0120                 if execHistory{blockIndex,i}==-1 % one of the necessary input has been deleted
0121                     % disp('*** The model and the selected block are valid, but at least one
necessary input has been deleted.')
0122                     UpdatePanel(3)
0123                     return;
0124                 end
0125             end
0126             % disp('*** The model and the selected block are valid, all necessary inputs
available.')
0127             UpdatePanel(4)
0128         end
0129     end
0130 else
0131     % disp('*** The model is not valid.')
0132     UpdatePanel(1)
0133 end
0134 end
0135
0136 %-----
0137
0138 %-----
0139 function UpdatePanel(varargin)
0140     % varargin{1}= 1 if all disabled (not a valid ltpda model or ltpda
block)
0141     %
0142     % 2 if the model is valid but the block is not.
0143     % 3 if the model and the block are valid, but the block
lacks the necessary inputs.
0144     % 4 model and block valid, all necessary inputs present.
0145     %
0146
0147     switch varargin{1}
0148     case 1
0149         set(findobj('UserData', 'currsystem'), 'String', 'None')
0150         set(findobj('UserData', 'currblock'), 'String', '----')
0151         set(findobj('Tag', 'partial1'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0152         set(findobj('Tag', 'partial2'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0153         set(findobj('Tag', 'partial3'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0154         set(findobj('Tag', 'infotext1'), 'ForegroundColor', [.8 .8 .8]);
0155         set(findobj('Tag', 'infotext2'), 'ForegroundColor', [.8 .8 .8]);
0156         set(findobj('Tag', 'infotext3'), 'ForegroundColor', [.8 .8 .8]);
0157     case 2
0158         set(findobj('UserData', 'currsystem'), 'String', bdroot)
0159         set(findobj('UserData', 'currblock'), 'String', 'Not valid')
0160         set(findobj('Tag', 'partial1'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0161         set(findobj('Tag', 'partial2'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0162         set(findobj('Tag', 'partial3'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0163         set(findobj('Tag', 'infotext1'), 'ForegroundColor', [.8 .8 .8]);
0164         set(findobj('Tag', 'infotext2'), 'ForegroundColor', [.8 .8 .8]);
0165         set(findobj('Tag', 'infotext3'), 'ForegroundColor', [.8 .8 .8]);
0166     case 3
0167         set(findobj('UserData', 'currsystem'), 'String', bdroot)
0168         set(findobj('UserData', 'currblock'), 'String', get_param(gcbbh, 'Name'))
0169         set(findobj('Tag', 'partial1'), 'CData', imread('iconpar1.jpg'), 'ButtonDownFcn', @Run1);
0170         set(findobj('Tag', 'partial2'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0171         set(findobj('Tag', 'partial3'), 'CData', imread('iconpar4.jpg'), 'ButtonDownFcn', '');
0172         set(findobj('Tag', 'infotext1'), 'ForegroundColor', [0 0 0]);
0173         set(findobj('Tag', 'infotext2'), 'ForegroundColor', [.8 .8 .8]);
0174         set(findobj('Tag', 'infotext3'), 'ForegroundColor', [.8 .8 .8]);
0175     case 4
0176         set(findobj('UserData', 'currsystem'), 'String', bdroot)
0177         set(findobj('UserData', 'currblock'), 'String', get_param(gcbbh, 'Name'))
0178         set(findobj('Tag', 'partial1'), 'CData', imread('iconpar1.jpg'), 'ButtonDownFcn', @Run1);
0179         set(findobj('Tag', 'partial2'), 'CData', imread('iconpar2.jpg'), 'ButtonDownFcn', @Run2);
0180         set(findobj('Tag', 'partial3'), 'CData', imread('iconpar3.jpg'), 'ButtonDownFcn', @Run3);
0181         set(findobj('Tag', 'infotext1'), 'ForegroundColor', [0 0 0]);
0182         set(findobj('Tag', 'infotext2'), 'ForegroundColor', [0 0 0]);
0183         set(findobj('Tag', 'infotext3'), 'ForegroundColor', [0 0 0]);
0184     end
0185
0186 end
0187
0188 %-----
0189
0190 %-----
0191 function Run1(varargin)

```

```

0191 % Execution from the beginning to the current block
0192
0193 % To find all parents, up to the beginning of the model:
0194     parentsBlock = unique(utls.prog.findparent(gcbh));
0195     parentsBlock = [parentsBlock ; find_system(gcbh, 'LookUnderMasks', 'all', 'BlockType', 'M-S-
Function')];];
0196
0197     assignin('base', 'executionList', parentsBlock);
0198     ltpdagui('redraw', 1.1);
0199
0200 end
0201 %-----
0202
0203 %-----
0204 function Run2(varargin)
0205 % Execution of the current block only
0206
0207     assignin('base', 'executionList', find_system(gcbh, 'LookUnderMasks', 'all', 'BlockType', 'M-S-
Function') );
0208     ltpdagui('redraw', 1.1);
0209
0210 end
0211 %-----
0212
0213 %-----
0214 function Run3(varargin)
0215 % Execution from the current block to the end
0216
0217 % To find all children, up to the end of the model:
0218     childrenBlock = unique(utls.prog.findchildren(gcbh));
0219     childrenBlock(:,2) = 0;
0220     while min(childrenBlock(:,2))~=0
0221         for i=1:size(childrenBlock,1)
0222             if childrenBlock(i,2)==0
0223                 nextGenerationChildren = utls.prog.findchildren(childrenBlock(i,1));
0224                 nextGenerationChildren(:,2) = 0;
0225                 if nextGenerationChildren(1,1)~=0
0226                     childrenBlock = [childrenBlock ; nextGenerationChildren];
0227                 end
0228                 childrenBlock(i,2) = 1;
0229             end
0230         end
0231         childrenBlock = unique(childrenBlock, 'rows');
0232     end
0233     childrenBlock(:,2)=[];
0234     childrenBlock = [gcbh ; childrenBlock];
0235
0236     assignin('base', 'executionList', childrenBlock);
0237     ltpdagui('redraw', 1.1);
0238
0239 end
0240 %-----
0241
0242 end

```

panXyettocome

PURPOSE ^

=====

SYNOPSIS ^

function varargout = panXyettocome(varargin)

DESCRIPTION ^

```
=====
===== Panel yet to come =====
=====
This is the GUI m-file to show that the relative panel (see ltpdagui.m)
is yet to be programmed.

$Id: panXyettocome.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdagui](#)

=====

SOURCE CODE ^

```
0001 function varargout = panXyettocome(varargin)
0002
0003 % =====
0004 % ===== Panel yet to come =====
0005 % =====
0006 % This is the GUI m-file to show that the relative panel (see ltpdagui.m)
0007 % is yet to be programmed.
0008 %
0009 % $Id: panXyettocome.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0010
0011 currPanel = varargin{1};
0012 panelDims = get(currPanel, 'Position');
0013 backColor = get(currPanel, 'BackgroundColor');
0014
0015 uicontrol(...
0016     'Parent',currPanel,...
0017     'BackgroundColor',backColor,...
0018     'Units','pixels',...
0019     'HorizontalAlignment','center',...
0020     'Position',[50, (panelDims(4)-30)/2, panelDims(3)-100, 30],...
0021     'Fontname','Times New Roman',...
0022     'FontSize',13,...
0023     'FontAngle','italic',...
0024     'String','This panel is yet to come, but it''s not intended for release in
v1.0',...
0025     'TooltipString','You ain''t seen anything yet',...
0026     'Visible','on',...
0027     'Style','text');
0028
```


progressBar

PURPOSE ^

Progress bar for the LTPDA GUI.

SYNOPSIS ^

function progressBar(varargin)

DESCRIPTION ^

```
Progress bar for the LTPDA GUI.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
This function is called whenever the user starts an analysis by clicking
on the 'Start' button in the LTPDA GUI.
It calculates how many blocks in the current Simulink model are
M-S-function blocks calling 'ltpdasim.m', ie. how many LTPDA analysis
blocks are present.
This excludes all 'service' blocks which do not perform any real
calculation, such as the input 'Object from list' block or the output
'Send to output' block: the execution of these blocks is almost immediate
and there would be non sense in considering them for the sake of the
'progress bar'.

Actually, the progress bar do NOT provide an estimate of the remaining
time, since this would be impossible to achieve (the time required by
every function/block is different and depending on its relative input,
so there's no way to estimate it). It only shows how many blocks there
are, how many have already been executed and how many to go.
Its primary purpose thus is provide the user with a perception of how the
analysis is going, and in case which block requires too much time to be
executed.

$Id: progressBar.m,v 1.5 2008/08/01 16:38:55 nicola Exp $
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [progressBar](#) Progress bar for the LTPDA GUI.

This function is called by:

-  [ltpdasim](#) This is the automatic function wrapper
-  [pan1main](#) This is the function for the main panel of the GUI LTPDA, the package of
-  [progressBar](#) Progress bar for the LTPDA GUI.

SOURCE CODE ^


```
0001 function progressBar(varargin)
0002 % Progress bar for the LTPDA GUI.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % This function is called whenever the user starts an analysis by clicking
0007 % on the 'Start' button in the LTPDA GUI.
0008 % It calculates how many blocks in the current Simulink model are
0009 % M-S-function blocks calling 'ltpdasim.m', ie. how many LTPDA analysis
0010 % blocks are present.
0011 % This excludes all 'service' blocks which do not perform any real
0012 % calculation, such as the input 'Object from list' block or the output
0013 % 'Send to output' block: the execution of these blocks is almost immediate
0014 % and there would be non sense in considering them for the sake of the
0015 % 'progress bar'.
0016 %
0017 % Actually, the progress bar do NOT provide an estimate of the remaining
0018 % time, since this would be impossible to achieve (the time required by
0019 % every function/block is different and depending on its relative input,
0020 % so there's no way to estimate it). It only shows how many blocks there
0021 % are, how many have already been executed and how many to go.
0022 % Its primary purpose thus is provide the user with a perception of how the
0023 % analysis is going, and in case which block requires too much time to be
0024 % executed.
0025 %
0026 % $Id: progressBar.m,v 1.5 2008/08/01 16:38:55 nicola Exp $
0027 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0028
0029     close(findobj('Name','LTPDA Progress Bar'));
0030
0031
0032     funcBlocks = utils.prog.find_in_models(bdroot,'LookUnderMasks','all','FunctionName','ltpdasim');
0033
0034     screenSize = get(0,'ScreenSize');
0035     backColor  = [0.98 , 0.98 , 0.98];
0036
0037     position   = [(screenSize(3)-400)/2,(screenSize(4)-200)/2,400,180];
0038     progressWind = figure('Position',position,'Name','LTPDA Progress
Bar','Tag','LTPDAProgressBar','Resize','off','NumberTitle','off','ToolBar','none','Menubar','none');
0039
0040     barBackSize = [position(3)-20,14];
0041     barBackDimension = [(position(3)-barBackSize(1))/2 , (position(4)-barBackSize(2))/2-25 , barBackSize];
0042     progressBack = axes('Parent',progressWind,'Units','pixels','Position',barBackDimension);
0043     image(imread('progressbarg.jpg'),'Parent',progressBack);
0044     axis off;
0045     barSize = [1,14];
0046     barDimension = [(position(3)-barBackSize(1))/2 , (position(4)-barBackSize(2))/2-25 , barSize];
0047     set(progressBack,'HandleVisibility','callback');
0048     progressBar = axes('Parent',progressWind,'Units','pixels','Position',barDimension);
0049     image(imread('progressbar.jpg'),'Parent',progressBar);
0050     set(progressBar,'Tag','progressaxes')
0051     set(progressBar,'UserData',barBackDimension)
0052     axis off;
0053     set(progressBar,'HandleVisibility','callback');
0054     set(gcf,'NextPlot','new');
0055
0056 % Total number of blocks
0057 uicontrol('Parent',progressWind,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[0 position(4)-40 position(3)
20],'String','Total number of blocks to compute:
',num2str(numel(funcBlocks)),'FontName','Arial','FontSize',9,'FontWeight','normal','Visible','on','Tag','blockstotal','UserData',numel(funcBlocks),'Style','
');
0058
0059 % Currently executed block
0060 uicontrol('Parent',progressWind,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[0 position(4)-70 position(3)
20],'String','','FontName','Arial','FontSize',9,'FontWeight','bold','Visible','on','Tag','currentexec','Style','text');
0061
0062 % Blocks done
0063 uicontrol('Parent',progressWind,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[10 (position(4)-barBackSize(2))/2-5 60
14],'String','Done: 0','FontName','Arial','FontSize',8,'FontWeight','normal','Visible','on','Tag','done','UserData',0,'Style','text');
0064
0065 % Blocks to go
0066 uicontrol('Parent',progressWind,'BackgroundColor',backColor,'HorizontalAlignment','right','Position',[position(3)-60-10 (position(4)-
barBackSize(2))/2-5 60 14],'String','To go:
',num2str(numel(funcBlocks)),'FontName','Arial','FontSize',8,'FontWeight','normal','Visible','on','Tag','togo','UserData',numel(funcBlocks),'Style','text'
```

Description of progressBar

```
);  
0067  
0068 % Press X to stop  
0069 uicontrol('Parent',progressWind,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[0 20 position(3) 20],'String','Press X to  
stop the analysis','FontName','Arial','FontSize',9,'FontWeight','normal','Visible','on','Style','text');  
0070  
0071 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

slblocks

PURPOSE ^

SLBLOCKS Defines the block library for a specific Toolbox or Blockset.

SYNOPSIS ^

function blkStruct = slblocks

DESCRIPTION ^

SLBLOCKS Defines the block library for a specific Toolbox or Blockset. SLBLOCKS returns information about a Blockset to Simulink. The information returned is in the form of a BlocksetStruct with the following fields:

Name	Name of the Blockset in the Simulink block library Blocksets & Toolboxes subsystem.
OpenFcn	MATLAB expression (function) to call when you double-click on the block in the Blocksets & Toolboxes subsystem.
MaskDisplay	Optional field that specifies the Mask Display commands to use for the block in the Blocksets & Toolboxes subsystem.
Browser	Array of Simulink Library Browser structures, described below.

The Simulink Library Browser needs to know which libraries in your Blockset it should show, and what names to give them. To provide this information, define an array of Browser data structures with one array element for each library to display in the Simulink Library Browser. Each array element has two fields:

Library	File name of the library (mdl-file) to include in the Library Browser.
Name	Name displayed for the library in the Library Browser window. Note that the Name is not required to be the same as the mdl-file name.

Example:

```
%
% Define the BlocksetStruct for the Simulink block libraries
% Only simulink_extras shows up in Blocksets & Toolboxes
%
blkStruct.Name          = ['Simulink' sprintf('\n') 'Extras'];
blkStruct.OpenFcn       = simulink_extras;
blkStruct.MaskDisplay   = disp('Simulink\nExtras');

%
% Both simulink and simulink_extras show up in the Library Browser.
%
blkStruct.Browser(1).Library = 'simulink';
blkStruct.Browser(1).Name   = 'Simulink';
blkStruct.Browser(2).Library = 'simulink_extras';
blkStruct.Browser(2).Name   = 'Simulink Extras';
```

See also FINDBLIB, LIBBROWSE.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```

0001 function blkStruct = slblocks
0002 %SLBLOCKS Defines the block library for a specific Toolbox or Blockset.
0003 % SLBLOCKS returns information about a Blockset to Simulink. The
0004 % information returned is in the form of a BlocksetStruct with the
0005 % following fields:
0006 %
0007 %     Name          Name of the Blockset in the Simulink block library
0008 %                   Blocksets & Toolboxes subsystem.
0009 %     OpenFcn       MATLAB expression (function) to call when you
0010 %                   double-click on the block in the Blocksets & Toolboxes
0011 %                   subsystem.
0012 %     MaskDisplay    Optional field that specifies the Mask Display commands
0013 %                   to use for the block in the Blocksets & Toolboxes
0014 %                   subsystem.
0015 %     Browser        Array of Simulink Library Browser structures, described
0016 %                   below.
0017 %
0018 % The Simulink Library Browser needs to know which libraries in your
0019 % Blockset it should show, and what names to give them. To provide
0020 % this information, define an array of Browser data structures with one
0021 % array element for each library to display in the Simulink Library
0022 % Browser. Each array element has two fields:
0023 %
0024 %     Library        File name of the library (mdl-file) to include in the
0025 %                   Library Browser.
0026 %     Name            Name displayed for the library in the Library Browser
0027 %                   window. Note that the Name is not required to be the
0028 %                   same as the mdl-file name.
0029 %
0030 % Example:
0031 %
0032 %     %
0033 %     % Define the BlocksetStruct for the Simulink block libraries
0034 %     % Only simulink_extras shows up in Blocksets & Toolboxes
0035 %     %
0036 %     blkStruct.Name          = ['Simulink' sprintf('\n') 'Extras'];
0037 %     blkStruct.OpenFcn       = simulink_extras;
0038 %     blkStruct.MaskDisplay = disp('Simulink\nExtras');
0039 %
0040 %     %
0041 %     % Both simulink and simulink_extras show up in the Library Browser.
0042 %     %
0043 %     blkStruct.Browser(1).Library = 'simulink';
0044 %     blkStruct.Browser(1).Name   = 'Simulink';
0045 %     blkStruct.Browser(2).Library = 'simulink_extras';
0046 %     blkStruct.Browser(2).Name   = 'Simulink Extras';
0047 %
0048 % See also FINDBLIB, LIBBROWSE.
0049 %
0050 % Copyright 1990-2006 The MathWorks, Inc.
0051 % $Id: slblocks.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0052 %
0053 %
0054 % Name of the subsystem which will show up in the Simulink Blocksets
0055 % and Toolboxes subsystem.
0056 %
0057 blkStruct.Name = ['LTPDA' sprintf('\n') 'Package'];
0058 %
0059 %
0060 % The function that will be called when the user double-clicks on
0061 % this icon.
0062 %
0063 blkStruct.OpenFcn = 'ltpdalib';
0064 %
0065 %
0066 % The argument to be set as the Mask Display for the subsystem. You
0067 % may comment this line out if no specific mask is desired.
0068 % Example: blkStruct.MaskDisplay = 'plot([0:2*pi],sin([0:2*pi]));';
0069 % No display for Simulink Extras.
0070 %
0071 blkStruct.MaskDisplay = '';
0072 %
0073 %
0074 % Define the Browser structure array, the first element contains the
0075 % information for the Simulink block library and the second for the
0076 % Simulink Extras block library.
0077 %
0078 % Browser(1).Library = 'simulink';
0079 % Browser(1).Name    = 'Simulink';
0080 % Browser(1).IsFlat  = 0;% Is this library "flat" (i.e. no subsystems)?
0081 %
0082 Browser(2).Library = 'ltpda_library';
0083 Browser(2).Name    = 'LTPDA Library';
0084 Browser(2).IsFlat  = 0;
0085 %
0086 blkStruct.Browser = Browser;

```

```
0087 clear Browser;
0088
0089 %
0090 % Define information about Signal Viewers
0091 %
0092 Viewer(1).Library = 'simviewers';
0093 Viewer(1).Name    = 'Simulink';
0094
0095 blkStruct.Viewer = Viewer;
0096 clear Viewer;
0097
0098 %
0099 % Define information about Signal Generators
0100 %
0101 Generator(1).Library = 'simgens';
0102 Generator(1).Name    = 'Simulink';
0103
0104 blkStruct.Generator = Generator;
0105 clear Generator;
0106
0107 % Define information for model updater
0108 %blkStruct.ModelUpdaterMethods.fhDetermineBrokenLinks =
@UpdateSimulinkBrokenLinksMappingHelper;
0109 blkStruct.ModelUpdaterMethods.fhUpdateModel = @UpdateSimulinkBlocksHelper;
0110
0111 % End of slblocks
0112
0113
```

uipickfiles

PURPOSE ^

uipickfiles: GUI program to select file(s) and/or directories.

SYNOPSIS ^

function out = uipickfiles(varargin)

DESCRIPTION ^

uipickfiles: GUI program to select file(s) and/or directories.

Syntax:
files = uipickfiles('PropertyName',PropertyValue,...)

The current directory can be changed by operating in the file navigator: double-clicking on a directory in the list to move further down the tree, using the popup menu to move up the tree, typing a path in the box to move to any directory or right-clicking on the path box to revisit a previously-listed directory.

Files can be added to the list by double-clicking or selecting files (non-contiguous selections are possible with the control key) and pressing the Add button. Files in the list can be removed or re-ordered. When finished, a press of the Done button will return the full paths to the selected files in a cell array, structure array or character array. If the Cancel button is pressed then zero is returned.

The following optional property/value pairs can be specified as arguments to control the indicated behavior:

Property	Value
FilterSpec	String to specify starting directory and/or file filter. Ex: 'C:\bin' will start up in that directory. '*.txt' will list only files ending in '.txt'. 'c:\bin*.txt' will do both. Default is to start up in the current directory and list all files. Can be changed with the GUI.
REFilter	String containing a regular expression used to filter the file list. Ex: '\.m\$.mat\$' will list files ending in '.m' and '.mat'. Default is empty string. Can be used with FilterSpec and both filters are applied. Can be changed with the GUI.
Prompt	String containing a prompt appearing in the title bar of the figure. Default is 'Select files'.
NumFiles	Scalar or vector specifying number of files that must be selected. A scalar specifies an exact value; a two-element vector can be used to specify a range, [min max]. The function will not return unless the specified number of files have been chosen. Default is [] which accepts any number of files.
Output	String specifying the data type of the output: 'cell', 'struct' or 'char'. Specifying 'cell' produces a cell array of strings, the strings containing the full paths of the chosen files. 'Struct' returns a structure array like the result of the dir function except that the 'name' field contains a full path instead of just the file name. 'Char' returns a character array of the full paths. This is most useful when you have just one file and want it in a string instead of a cell array containing just one string. The default is 'cell'.

All properties and values are case-insensitive and need only be unambiguous. For example,

```
files = uipickfiles('num',1,'out','ch')
```

is valid usage.

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

-  [pan1main](#) This is the function for the main panel of the GUI LTPDA, the package of
-  [pan2param](#) This is the GUI m-file able to read the list of parameters to be set for

SUBFUNCTIONS [^]

-  [function add\(varargin\)](#)
-  [function remove\(varargin\)](#)
-  [function open\(varargin\)](#)
-  [function clicknav\(varargin\)](#)
-  [function clickpicks\(varargin\)](#)
-  [function dirpopup\(varargin\)](#)
-  [function change_path\(varargin\)](#)
-  [function showfullpath\(varargin\)](#)
-  [function removedupes\(varargin\)](#)
-  [function moveup\(varargin\)](#)
-  [function movedown\(varargin\)](#)
-  [function togglefilter\(varargin\)](#)
-  [function setfilspec\(varargin\)](#)
-  [function setrefilter\(varargin\)](#)
-  [function done\(varargin\)](#)
-  [function cancel\(varargin\)](#)
-  [function history_cb\(varargin\)](#)
-  [function c = path2cell\(p\)](#)
-  [function p = cell2path\(c\)](#)
-  [function d = filtered_dir\(full_filter,re_filter\)](#)
-  [function drives = getdrives](#)
-  [function filenames = annotate_file_names\(filenames,dir_listing\)](#)
-  [function hist_menus = make_history_cm\(cb,hist_cm,hist_menus,history\)](#)

SOURCE CODE [^]

```

0001 function out = uipickfiles(varargin)
0002 %uipickfiles: GUI program to select file(s) and/or directories.
0003 %
0004 % Syntax:
0005 %   files = uipickfiles('PropertyName',PropertyValue,...)
0006 %
0007 % The current directory can be changed by operating in the file navigator:
0008 % double-clicking on a directory in the list to move further down the tree,
0009 % using the popup menu to move up the tree, typing a path in the box to
0010 % move to any directory or right-clicking on the path box to revisit a
0011 % previously-listed directory.
0012 %
0013 % Files can be added to the list by double-clicking or selecting files
0014 % (non-contiguous selections are possible with the control key) and
0015 % pressing the Add button. Files in the list can be removed or re-ordered.
0016 % When finished, a press of the Done button will return the full paths to
0017 % the selected files in a cell array, structure array or character array.
0018 % If the Cancel button is pressed then zero is returned.
0019 %
0020 % The following optional property/value pairs can be specified as arguments
0021 % to control the indicated behavior:
0022 %
0023 %   Property      Value
0024 %   -----

```



```

0025 % FilterSpec String to specify starting directory and/or file filter.
0026 % Ex: 'C:\bin' will start up in that directory. '*.txt'
0027 % will list only files ending in '.txt'. 'c:\bin\*.txt' will
0028 % do both. Default is to start up in the current directory
0029 % and list all files. Can be changed with the GUI.
0030 %
0031 % REFilter String containing a regular expression used to filter the
0032 % file list. Ex: '\.m$|\.mat$' will list files ending in
0033 % '.m' and '.mat'. Default is empty string. Can be used
0034 % with FilterSpec and both filters are applied. Can be
0035 % changed with the GUI.
0036 %
0037 % Prompt String containing a prompt appearing in the title bar of
0038 % the figure. Default is 'Select files'.
0039 %
0040 % NumFiles Scalar or vector specifying number of files that must be
0041 % selected. A scalar specifies an exact value; a two-element
0042 % vector can be used to specify a range, [min max]. The
0043 % function will not return unless the specified number of
0044 % files have been chosen. Default is [] which accepts any
0045 % number of files.
0046 %
0047 % Output String specifying the data type of the output: 'cell',
0048 % 'struct' or 'char'. Specifying 'cell' produces a cell
0049 % array of strings, the strings containing the full paths of
0050 % the chosen files. 'Struct' returns a structure array like
0051 % the result of the dir function except that the 'name' field
0052 % contains a full path instead of just the file name. 'Char'
0053 % returns a character array of the full paths. This is most
0054 % useful when you have just one file and want it in a string
0055 % instead of a cell array containing just one string. The
0056 % default is 'cell'.
0057 %
0058 % All properties and values are case-insensitive and need only be
0059 % unambiguous. For example,
0060 %
0061 % files = uipickfiles('num',1,'out','ch')
0062 %
0063 % is valid usage.
0064 %
0065 % Version: 1.0, 25 April 2006
0066 % Author: Douglas M. Schwarz
0067 % Email: dmschwarz=ieee*org, dmschwarz=urgrad*rochester*edu
0068 % Real_email = regexp(Email,{'=','*'},{'@','.'})
0069 % $Id: uipickfiles.m,v 1.1 2008/03/01 13:43:20 nicola Exp $
0070
0071
0072 % Define properties and set default values.
0073 prop.filterspec = '*';
0074 prop.refilter = '';
0075 prop.prompt = 'Select files';
0076 prop.numfiles = [];
0077 prop.output = 'cell';
0078
0079 % Process inputs and set prop fields.
0080 properties = fieldnames(prop);
0081 arg_index = 1;
0082 while arg_index <= nargin
0083     arg = varargin{arg_index};
0084     if ischar(arg)
0085         prop_index = find(strncmpi(arg,properties,length(arg)));
0086         if length(prop_index) == 1
0087             prop.(properties{prop_index}) = varargin{arg_index + 1};
0088         else
0089             error('Property '%s'' does not exist or is ambiguous.',arg)
0090         end
0091         arg_index = arg_index + 2;
0092     elseif isstruct(arg)
0093         arg_fn = fieldnames(arg);
0094         for i = 1:length(arg_fn)
0095             prop_index = find(strncmpi(arg_fn{i},properties,...
0096                 length(arg_fn{i})));
0097             if length(prop_index) == 1
0098                 prop.(properties{prop_index}) = arg.(arg_fn{i});
0099             else
0100                 error('Property '%s'' does not exist or is ambiguous.',...
0101                     arg_fn{i})
0102             end
0103         end
0104         arg_index = arg_index + 1;
0105     else
0106         error(['Properties must be specified by property/value pairs',...
0107             ' or structures.'])
0108     end
0109 end
0110

```



```

0111 % Validate FilterSpec property.
0112 if isempty(prop.filterspec)
0113     prop.filterspec = '*';
0114 end
0115 if ~ischar(prop.filterspec)
0116     error('FilterSpec property must contain a string.')
0117 end
0118
0119 % Validate REFilter property.
0120 if ~ischar(prop.refilter)
0121     error('REFilter property must contain a string.')
0122 end
0123
0124 % Validate Prompt property.
0125 if ~ischar(prop.prompt)
0126     error('Prompt property must contain a string.')
0127 end
0128
0129 % Validate NumFiles property.
0130 if numel(prop.numfiles) > 2 || any(prop.numfiles < 0)
0131     error('NumFiles must be empty, a scalar or two-element vector.')
0132 end
0133 prop.numfiles = unique(prop.numfiles);
0134 if isequal(prop.numfiles,1)
0135     numstr = 'Select exactly 1 file.';
0136 elseif length(prop.numfiles) == 1
0137     numstr = sprintf('Select exactly %d files.',prop.numfiles);
0138 else
0139     numstr = sprintf('Select %d to %d files.',prop.numfiles);
0140 end
0141
0142 % Validate Output property.
0143 legal_outputs = {'cell','struct','char'};
0144 out_idx = find(strncmpi(prop.output,legal_outputs,length(prop.output)));
0145 if length(out_idx) == 1
0146     prop.output = legal_outputs{out_idx};
0147 else
0148     error(['Value of ''Output'' property, ''%s'', is illegal or ''...
0149           'ambiguous.'],prop.output)
0150 end
0151
0152
0153 % Initialize file lists.
0154 [current_dir,f,e] = fileparts(prop.filterspec);
0155 filter = [f,e];
0156 if isempty(current_dir)
0157     current_dir = pwd;
0158 end
0159 if isempty(filter)
0160     filter = '*';
0161 end
0162 re_filter = prop.refilter;
0163 full_filter = fullfile(current_dir,filter);
0164 path_cell = path2cell(current_dir);
0165 fdir = filtered_dir(full_filter,re_filter);
0166 filenames = {fdir.name};
0167 filenames = annotate_file_names(filenames,fdir);
0168
0169 % Initialize some data.
0170 file_picks = {};
0171 full_file_picks = {};
0172 dir_picks = dir(' '); % Create empty directory structure.
0173 show_full_path = false;
0174 nodupes = true;
0175 history = {current_dir};
0176
0177 % Create figure.
0178 gray = get(0,'DefaultUIControlBackgroundColor');
0179 fig = figure('Position',[0 0 740 445],...
0180     'Color',gray,...
0181     'WindowStyle','modal',...
0182     'Resize','off',...
0183     'NumberTitle','off',...
0184     'Name',prop.prompt,...
0185     'IntegerHandle','off',...
0186     'CloseRequestFcn',@cancel,...
0187     'CreateFcn',{@movegui,'center'});
0188
0189 % Create uicontrols.
0190 uicontrol('Style','frame',...
0191     'Position',[255 260 110 70])
0192 uicontrol('Style','frame',...
0193     'Position',[275 135 110 100])
0194
0195 navlist = uicontrol('Style','listbox',...
0196     'Position',[10 10 250 320],...

```

```

0197     'String', filenames, ...
0198     'Value', [], ...
0199     'BackgroundColor', 'w', ...
0200     'Callback', @clicknav, ...
0201     'Max', 2);
0202 pickslist = uicontrol('Style','listbox',...
0203     'Position',[380 10 350 320],...
0204     'String', {}, ...
0205     'BackgroundColor', 'w', ...
0206     'Callback', @clickpicks, ...
0207     'Max', 2);
0208
0209 openbut = uicontrol('Style','pushbutton',...
0210     'Position',[270 300 80 20],...
0211     'String','Open',...
0212     'Enable','off',...
0213     'Callback',@open);
0214 arrow = [2 2 2 2 2 2 2 2 2 1 2 2 2;...
0215         2 2 2 2 2 2 2 2 2 0 2 2;...
0216         2 2 2 2 2 2 2 2 2 2 0 2;...
0217         0 0 0 0 0 0 0 0 0 0 0 0;...
0218         2 2 2 2 2 2 2 2 2 2 0 2;...
0219         2 2 2 2 2 2 2 2 2 0 2 2;...
0220         2 2 2 2 2 2 2 2 2 1 2 2];
0221 arrow(arrow == 2) = NaN;
0222 arrow_im = NaN*ones(16,76);
0223 arrow_im(6:12,45:56) = arrow/2;
0224 im = repmat(arrow_im,[1 1 3]);
0225 addbut = uicontrol('Style','pushbutton',...
0226     'Position',[270 270 80 20],...
0227     'String','Add',...
0228     'Enable','off',...
0229     'CData',im,...
0230     'Callback',@add);
0231
0232 removebut = uicontrol('Style','pushbutton',...
0233     'Position',[290 205 80 20],...
0234     'String','Remove',...
0235     'Enable','off',...
0236     'Callback',@remove);
0237 moveupbut = uicontrol('Style','pushbutton',...
0238     'Position',[290 175 80 20],...
0239     'String','Move Up',...
0240     'Enable','off',...
0241     'Callback',@moveup);
0242 movedownbut = uicontrol('Style','pushbutton',...
0243     'Position',[290 145 80 20],...
0244     'String','Move Down',...
0245     'Enable','off',...
0246     'Callback',@movedown);
0247
0248 uicontrol('Position',[10 380 250 16],...
0249     'Style','text',...
0250     'String','Current Directory',...
0251     'HorizontalAlignment','center')
0252 dir_popup = uicontrol('Style','popupmenu',...
0253     'Position',[10 335 250 20],...
0254     'BackgroundColor','w',...
0255     'String',path_cell(end:-1:1),...
0256     'Value',1,...
0257     'Callback',@dirpopup);
0258 hist_cm = uicontextmenu;
0259 pathbox = uicontrol('Position',[10 360 250 20],...
0260     'Style','edit',...
0261     'BackgroundColor','w',...
0262     'String',current_dir,...
0263     'HorizontalAlignment','left',...
0264     'Callback',@change_path,...
0265     'UIContextMenu',hist_cm);
0266 hist_menus = [];
0267 hist_cb = @history_cb;
0268 hist_menus = make_history_cm(hist_cb,hist_cm,hist_menus,history);
0269
0270 uicontrol('Position',[10 425 80 16],...
0271     'Style','text',...
0272     'String','File Filter',...
0273     'HorizontalAlignment','left')
0274 uicontrol('Position',[100 425 160 16],...
0275     'Style','text',...
0276     'String','Reg. Exp. Filter',...
0277     'HorizontalAlignment','left')
0278 showallfiles = uicontrol('Position',[270 405 100 20],...
0279     'Style','checkbox',...
0280     'String','Show All Files',...
0281     'Value',0,...
0282     'HorizontalAlignment','left',...

```

```

0283     'Callback',@togglefilter);
0284 filter_ed = uicontrol('Position',[10 405 80 20],...
0285     'Style','edit',...
0286     'BackgroundColor','w',...
0287     'String',filter,...
0288     'HorizontalAlignment','left',...
0289     'Callback',@setfilspec);
0290 refilter_ed = uicontrol('Position',[100 405 160 20],...
0291     'Style','edit',...
0292     'BackgroundColor','w',...
0293     'String',re_filter,...
0294     'HorizontalAlignment','left',...
0295     'Callback',@setrefilter);
0296
0297 viewfullpath = uicontrol('Style','checkbox',...
0298     'Position',[380 335 230 20],...
0299     'String','Show full paths',...
0300     'Value',show_full_path,...
0301     'HorizontalAlignment','left',...
0302     'Callback',@showfullpath);
0303 remove_dupes = uicontrol('Style','checkbox',...
0304     'Position',[380 360 230 20],...
0305     'String','Remove duplicates (as per full path)',...
0306     'Value',nodupes,...
0307     'HorizontalAlignment','left',...
0308     'Callback',@removedupes);
0309 uicontrol('Position',[380 405 350 20],...
0310     'Style','text',...
0311     'String','Selected Files',...
0312     'HorizontalAlignment','center')
0313 uicontrol('Position',[280 80 80 30], 'String','Done',...
0314     'Callback',@done);
0315 uicontrol('Position',[280 30 80 30], 'String','Cancel',...
0316     'Callback',@cancel);
0317
0318 if ~isempty(prop.numfiles)
0319     uicontrol('Position',[380 385 350 16],...
0320         'Style','text',...
0321         'String',numstr,...
0322         'ForegroundColor','r',...
0323         'HorizontalAlignment','center')
0324 end
0325
0326 set(fig,'HandleVisibility','off')
0327
0328 uiwait(fig)
0329
0330 % Compute desired output.
0331 switch prop.output
0332     case 'cell'
0333         out = full_file_picks;
0334     case 'struct'
0335         out = dir_picks(:);
0336     case 'char'
0337         out = char(full_file_picks);
0338     case 'cancel'
0339         out = 0;
0340 end
0341
0342 % ----- Callback functions -----
0343
0344 function add(varargin)
0345     values = get(navlist,'Value');
0346     for i = 1:length(values)
0347         dir_pick = fdir(values(i));
0348         pick = dir_pick.name;
0349         pick_full = fullfile(current_dir,pick);
0350         dir_pick.name = pick_full;
0351         if ~nodupes || ~any(strcmp(full_file_picks,pick_full))
0352             file_picks{end + 1} = pick;
0353             full_file_picks{end + 1} = pick_full;
0354             dir_picks(end + 1) = dir_pick;
0355         end
0356     end
0357     if show_full_path
0358         set(pickslist, 'String',full_file_picks, 'Value', []);
0359     else
0360         set(pickslist, 'String',file_picks, 'Value', []);
0361     end
0362     set([removebut,moveupbut,movedownbut], 'Enable', 'off');
0363 end
0364
0365 function remove(varargin)
0366     values = get(pickslist, 'Value');
0367     file_picks(values) = [];
0368     full_file_picks(values) = [];

```

```

0369     dir_picks(values) = [];
0370     top = get(pickslist, 'ListboxTop');
0371     num_above_top = sum(values < top);
0372     top = top - num_above_top;
0373     num_picks = length(file_picks);
0374     new_value = min(min(values) - num_above_top, num_picks);
0375     if num_picks == 0
0376         new_value = [];
0377         set([removebut, moveupbut, movedownbut], 'Enable', 'off')
0378     end
0379     if show_full_path
0380         set(pickslist, 'String', full_file_picks, 'Value', new_value, ...
0381             'ListboxTop', top)
0382     else
0383         set(pickslist, 'String', file_picks, 'Value', new_value, ...
0384             'ListboxTop', top)
0385     end
0386 end
0387
0388 function open(varargin)
0389     values = get(navlist, 'Value');
0390     if fdir(values).isdir
0391         if strcmp(fdir(values).name, '.')
0392             return
0393         elseif strcmp(fdir(values).name, '..')
0394             set(dir_popup, 'Value', min(2, length(path_cell)))
0395             dirpopup();
0396             return
0397         end
0398         current_dir = fullfile(current_dir, fdir(values).name);
0399         history{end+1} = current_dir;
0400         history = unique(history);
0401         hist_menus = make_history_cm(hist_cb, hist_cm, hist_menus, ...
0402             history);
0403         full_filter = fullfile(current_dir, filter);
0404         path_cell = path2cell(current_dir);
0405         fdir = filtered_dir(full_filter, re_filter);
0406         filenames = {fdir.name}';
0407         filenames = annotate_file_names(filenames, fdir);
0408         set(dir_popup, 'String', path_cell(end:-1:1), 'Value', 1)
0409         set(pathbox, 'String', current_dir)
0410         set(navlist, 'ListboxTop', 1, 'Value', [], 'String', filenames)
0411         set(addbut, 'Enable', 'off')
0412         set(openbut, 'Enable', 'off')
0413     end
0414 end
0415
0416 function clicknav(varargin)
0417     value = get(navlist, 'Value');
0418     nval = length(value);
0419     dbl_click_fcn = @add;
0420     switch nval
0421     case 0
0422         set([addbut, openbut], 'Enable', 'off')
0423     case 1
0424         set(addbut, 'Enable', 'on');
0425         if fdir(value).isdir
0426             set(openbut, 'Enable', 'on')
0427             dbl_click_fcn = @open;
0428         else
0429             set(openbut, 'Enable', 'off')
0430         end
0431     otherwise
0432         set(addbut, 'Enable', 'on')
0433         set(openbut, 'Enable', 'off')
0434     end
0435     if strcmp(get(fig, 'SelectionType'), 'open')
0436         dbl_click_fcn();
0437     end
0438 end
0439
0440 function clickpicks(varargin)
0441     value = get(pickslist, 'Value');
0442     if isempty(value)
0443         set([removebut, moveupbut, movedownbut], 'Enable', 'off')
0444     else
0445         set(removebut, 'Enable', 'on')
0446         if min(value) == 1
0447             set(moveupbut, 'Enable', 'off')
0448         else
0449             set(moveupbut, 'Enable', 'on')
0450         end
0451         if max(value) == length(file_picks)
0452             set(movedownbut, 'Enable', 'off')
0453         else
0454             set(movedownbut, 'Enable', 'on')
0455         end
0456     end
0457 end

```

```

0455         end
0456     end
0457     if strcmp(get(fig, 'SelectionType'), 'open')
0458         remove();
0459     end
0460 end
0461
0462 function dirpopup(varargin)
0463     value = get(dir_popup, 'Value');
0464     len = length(path_cell);
0465     path_cell = path_cell(1:end-value+1);
0466     if ispc && value == len
0467         current_dir = '';
0468         full_filter = filter;
0469         fdir = struct('name', getdrives, 'date', datestr(now), ...
0470             'bytes', 0, 'isdir', 1);
0471     else
0472         current_dir = cell2path(path_cell);
0473         history{end+1} = current_dir;
0474         history = unique(history);
0475         hist_menus = make_history_cm(hist_cb, hist_cm, hist_menus, ...
0476             history);
0477         full_filter = fullfile(current_dir, filter);
0478         fdir = filtered_dir(full_filter, re_filter);
0479     end
0480     filenames = {fdir.name}';
0481     filenames = annotate_file_names(filenames, fdir);
0482     set(dir_popup, 'String', path_cell(end:-1:1), 'Value', 1)
0483     set(pathbox, 'String', current_dir)
0484     set(navlist, 'String', filenames, 'Value', [])
0485     set(addbut, 'Enable', 'off')
0486 end
0487
0488 function change_path(varargin)
0489     proposed_path = get(pathbox, 'String');
0490     % Process any directories named '...'
0491     proposed_path_cell = path2cell(proposed_path);
0492     ddots = strcmp(proposed_path_cell, '.');
0493     ddots(find(ddots) - 1) = true;
0494     proposed_path_cell(ddots) = [];
0495     proposed_path = cell2path(proposed_path_cell);
0496     % Check for existance of directory.
0497     if ~exist(proposed_path, 'dir')
0498         uiwait(errordlg(['Directory "', proposed_path, ...
0499             '" does not exist.'], '', 'modal'))
0500     end
0501     return
0502     current_dir = proposed_path;
0503     history{end+1} = current_dir;
0504     history = unique(history);
0505     hist_menus = make_history_cm(hist_cb, hist_cm, hist_menus, history);
0506     full_filter = fullfile(current_dir, filter);
0507     path_cell = path2cell(current_dir);
0508     fdir = filtered_dir(full_filter, re_filter);
0509     filenames = {fdir.name}';
0510     filenames = annotate_file_names(filenames, fdir);
0511     set(dir_popup, 'String', path_cell(end:-1:1), 'Value', 1)
0512     set(pathbox, 'String', current_dir)
0513     set(navlist, 'String', filenames, 'Value', [])
0514     set(addbut, 'Enable', 'off')
0515     set(openbut, 'Enable', 'off')
0516 end
0517
0518 function showfullpath(varargin)
0519     show_full_path = get(viewfullpath, 'Value');
0520     if show_full_path
0521         set(pickslist, 'String', full_file_picks)
0522     else
0523         set(pickslist, 'String', file_picks)
0524     end
0525 end
0526
0527 function removedupes(varargin)
0528     nodupes = get(remove_dupes, 'Value');
0529     if nodupes
0530         num_picks = length(full_file_picks);
0531         [unused, rev_order] = unique(full_file_picks(end:-1:1));
0532         order = sort(num_picks + 1 - rev_order);
0533         full_file_picks = full_file_picks(order);
0534         file_picks = file_picks(order);
0535         if show_full_path
0536             set(pickslist, 'String', full_file_picks, 'Value', [])
0537         else
0538             set(pickslist, 'String', file_picks, 'Value', [])
0539         end
0540     end
    set([removebut, moveupbut, movedownbut], 'Enable', 'off')

```



```

0541     end
0542 end
0543
0544 function moveup(varargin)
0545     value = get(pickslist, 'Value');
0546     set(removebut, 'Enable', 'on')
0547     n = length(file_picks);
0548     omega = 1:n;
0549     index = zeros(1,n);
0550     index(value - 1) = omega(value);
0551     index(setdiff(omega,value - 1)) = omega(setdiff(omega,value));
0552     file_picks = file_picks(index);
0553     full_file_picks = full_file_picks(index);
0554     value = value - 1;
0555     if show_full_path
0556         set(pickslist, 'String', full_file_picks, 'Value', value)
0557     else
0558         set(pickslist, 'String', file_picks, 'Value', value)
0559     end
0560     if min(value) == 1
0561         set(moveupbut, 'Enable', 'off')
0562     end
0563     set(movedownbut, 'Enable', 'on')
0564 end
0565
0566 function movedown(varargin)
0567     value = get(pickslist, 'Value');
0568     set(removebut, 'Enable', 'on')
0569     n = length(file_picks);
0570     omega = 1:n;
0571     index = zeros(1,n);
0572     index(value + 1) = omega(value);
0573     index(setdiff(omega,value + 1)) = omega(setdiff(omega,value));
0574     file_picks = file_picks(index);
0575     full_file_picks = full_file_picks(index);
0576     value = value + 1;
0577     if show_full_path
0578         set(pickslist, 'String', full_file_picks, 'Value', value)
0579     else
0580         set(pickslist, 'String', file_picks, 'Value', value)
0581     end
0582     if max(value) == n
0583         set(movedownbut, 'Enable', 'off')
0584     end
0585     set(moveupbut, 'Enable', 'on')
0586 end
0587
0588 function togglefilter(varargin)
0589     value = get(showallfiles, 'Value');
0590     if value
0591         filter = '*';
0592         re_filter = '';
0593         set([filter_ed, refilter_ed], 'Enable', 'off')
0594     else
0595         filter = get(filter_ed, 'String');
0596         re_filter = get(refilter_ed, 'String');
0597         set([filter_ed, refilter_ed], 'Enable', 'on')
0598     end
0599     full_filter = fullfile(current_dir, filter);
0600     fdir = filtered_dir(full_filter, re_filter);
0601     filenames = {fdir.name}';
0602     filenames = annotate_file_names(filenames, fdir);
0603     set(navlist, 'String', filenames, 'Value', [])
0604     set(addbut, 'Enable', 'off')
0605 end
0606
0607 function setfilspec(varargin)
0608     filter = get(filter_ed, 'String');
0609     if isempty(filter)
0610         filter = '*';
0611         set(filter_ed, 'String', filter)
0612     end
0613     % Process file spec if a subdirectory was included.
0614     [p,f,e] = fileparts(filter);
0615     if ~isempty(p)
0616         newpath = fullfile(current_dir, p, '');
0617         set(pathbox, 'String', newpath)
0618         filter = [f,e];
0619         if isempty(filter)
0620             filter = '*';
0621         end
0622         set(filter_ed, 'String', filter)
0623         change_path();
0624     end
0625     full_filter = fullfile(current_dir, filter);
0626     fdir = filtered_dir(full_filter, re_filter);

```

```

0627     filenames = {fdir.name}';
0628     filenames = annotate_file_names(filenames,fdir);
0629     set(navlist,'String',filenames,'Value',[])
0630     set(addbut,'Enable','off')
0631 end
0632
0633 function setrefilter(varargin)
0634     re_filter = get(refilter_ed,'String');
0635     fdir = filtered_dir(full_filter,re_filter);
0636     filenames = {fdir.name}';
0637     filenames = annotate_file_names(filenames,fdir);
0638     set(navlist,'String',filenames,'Value',[])
0639     set(addbut,'Enable','off')
0640 end
0641
0642 function done(varargin)
0643     % Optional shortcut: click on a file and press 'Done'.
0644     % if isempty(full_file_picks) && strcmp(get(addbut,'Enable'),'on')
0645     %     add();
0646     % end
0647     numfiles = length(full_file_picks);
0648     if ~isempty(prop.numfiles)
0649         if numfiles < prop.numfiles(1)
0650             msg = {'Too few files selected.',numstr};
0651             uiwait(errordlg(msg,'','modal'))
0652             return
0653         elseif numfiles > prop.numfiles(end)
0654             msg = {'Too many files selected.',numstr};
0655             uiwait(errordlg(msg,'','modal'))
0656             return
0657         end
0658     end
0659     delete(fig)
0660 end
0661
0662 function cancel(varargin)
0663     prop.output = 'cancel';
0664     delete(fig)
0665 end
0666
0667 function history_cb(varargin)
0668     current_dir = history{varargin{3}};
0669     full_filter = fullfile(current_dir,filter);
0670     path_cell = path2cell(current_dir);
0671     fdir = filtered_dir(full_filter,re_filter);
0672     filenames = {fdir.name}';
0673     filenames = annotate_file_names(filenames,fdir);
0674     set(dir_popup,'String',path_cell(end:-1:1),'Value',1)
0675     set(pathbox,'String',current_dir)
0676     set(navlist,'ListboxTop',1,'Value',[],'String',filenames)
0677     set(addbut,'Enable','off')
0678     set(openbut,'Enable','off')
0679 end
0680 end
0681
0682
0683 % ----- Subfunctions -----
0684
0685 function c = path2cell(p)
0686 % Turns a path string into a cell array of path elements.
0687 c = strread(p,'%s','delimiter','\\');
0688 if ispc
0689     c = [{'My Computer'};c];
0690 else
0691     c = [{filesep};c(2:end)];
0692 end
0693 end
0694
0695
0696 function p = cell2path(c)
0697 % Turns a cell array of path elements into a path string.
0698 if ispc
0699     p = fullfile(c{2:end}, '');
0700 else
0701     p = fullfile(c{:}, '');
0702 end
0703 end
0704
0705
0706 function d = filtered_dir(full_filter,re_filter)
0707 % Like dir, but applies filters and sorting.
0708 p = fileparts(full_filter);
0709 if isempty(p) && full_filter(1) == '/'
0710     p = '/';
0711 end
0712 if exist(full_filter,'dir')

```

```

0713     c = cell(0,1);
0714     dfiles = struct('name',c,'date',c,'bytes',c,'isdir',c);
0715 else
0716     dfiles = dir(full_filter);
0717 end
0718 if ~isempty(dfiles)
0719     dfiles([dfiles.isdir]) = [];
0720 end
0721 ddir = dir(p);
0722 ddir = ddir([ddir.isdir]);
0723 % Additional regular expression filter.
0724 if nargin > 1 && ~isempty(re_filter)
0725     if ispc
0726         no_match = cellfun('isempty',regexpi({dfiles.name},re_filter));
0727     else
0728         no_match = cellfun('isempty',regexp({dfiles.name},re_filter));
0729     end
0730     dfiles(no_match) = [];
0731 end
0732 % Set navigator style:
0733 %     1 => mix file and directory names
0734 %     2 => means list all files before all directories
0735 %     3 => means list all directories before all files
0736 %     4 => same as 2 except put . and .. directories first
0737 if isunix
0738     style = 4;
0739 else
0740     style = 4;
0741 end
0742 switch style
0743 case 1
0744     d = [dfiles;ddir];
0745     [unused,index] = sort({d.name});
0746     d = d(index);
0747 case 2
0748     [unused,index1] = sort({dfiles.name});
0749     [unused,index2] = sort({ddir.name});
0750     d = [dfiles(index1);ddir(index2)];
0751 case 3
0752     [unused,index1] = sort({dfiles.name});
0753     [unused,index2] = sort({ddir.name});
0754     d = [ddir(index2);dfiles(index1)];
0755 case 4
0756     [unused,index1] = sort({dfiles.name});
0757     dot1 = find(strcmp({ddir.name}, '.'));
0758     dot2 = find(strcmp({ddir.name}, '..'));
0759     ddot1 = ddir(dot1);
0760     ddot2 = ddir(dot2);
0761     ddir([dot1,dot2]) = [];
0762     [unused,index2] = sort({ddir.name});
0763     d = [ddot1;ddot2;dfiles(index1);ddir(index2)];
0764 end
0765 end
0766
0767 function drives = getdrives
0768 % Returns a cell array of drive names on Windows.
0769 letters = char('A':'Z');
0770 num_letters = length(letters);
0771 drives = cell(1,num_letters);
0772 for i = 1:num_letters
0773     if exist([letters(i),':\'],'dir');
0774         drives{i} = [letters(i),':'];
0775     end
0776 end
0777 drives(cellfun('isempty',drives)) = [];
0778 end
0779
0780
0781 function filenames = annotate_file_names(filenames,dir_listing)
0782 % Adds a trailing filesep character to directory names.
0783 fs = filesep;
0784 for i = 1:length(filenames)
0785     if dir_listing(i).isdir
0786         filenames{i} = [filenames{i},fs];
0787     end
0788 end
0789 end
0790 end
0791
0792
0793 function hist_menus = make_history_cm(cb,hist_cm,hist_menus,history)
0794 % Make context menu for history.
0795 if ~isempty(hist_menus)
0796     delete(hist_menus)
0797 end
0798 num_hist = length(history);

```



```
0799 hist_menus = zeros(1,num_hist);
0800 for i = 1:num_hist
0801     hist_menus(i) = uimenu(hist_cm, 'Label', history{i}, ...
0802         'Callback', {cb,i});
0803 end
0804 end
```

welcome2LTPDA

PURPOSE ^

Introductory window for the LTPDA Toolbox.

SYNOPSIS ^

function welcome2LTPDA(varargin)

DESCRIPTION ^

Introductory window for the LTPDA Toolbox.

The animated GIF code is taken from 'gifplayer.m' by
Vihang Patil, Oct 2006
Copyright 2006-2007 Vihang Patil
Email: vihang_patil@yahoo.com
Created: 17th Oct 2006

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 loginLTPDA LOGINDLG Dialog for visually secure login.

SUBFUNCTIONS ^

-  function TmrFcn(src,event,handles)
-  function DeleteFunction(src,event,handles)

SOURCE CODE ^

```
0001 function welcome2LTPDA(varargin)
0002 % Introductory window for the LTPDA Toolbox.
0003 %
0004 % The animated GIF code is taken from 'gifplayer.m' by
0005 %     Vihang Patil, Oct 2006
0006 %     Copyright 2006-2007 Vihang Patil
0007 %     Email: vihang_patil@yahoo.com
0008 %     Created: 17th Oct 2006
0009
0010 global timePeriod
0011
0012 timePeriod = 0;
0013 screenSize = get(0, 'ScreenSize');
0014
0015 try
0016     guiName = varargin{1}; guiSize = varargin{2};
0017 catch
0018     guiName = ''; guiSize = [1 1 450 500];
0019 end
0020
0021 introSize = [(screenSize(3)-guiSize(3))/2 , (screenSize(4)-guiSize(4))/2 , guiSize(3) , guiSize(4)];
0022 introFig =
figure('Position',introSize,'Resize','off','Name',guiName,'NumberTitle','off','ToolBar','none','Menubar','none','Tag',
);
0023
0024
0025 logoSize = [303,303];
0026 introPosition = [(introSize(3)-logoSize(1))/2 , (introSize(4)-logoSize(2))/2 , logoSize];
0027 logo = axes('Parent',introFig,'Units','pixels','Position',introPosition);
0028 image(imread('ltp.jpg'),'Parent',logo,'AlphaData',imread('ltpalpha.gif'));
0029 axis(logo,'off'); axis(logo,'square');
0030 set(logo,'HandleVisibility','callback');
0031
0032 uicontrol('Parent',introFig,'BackgroundColor',[1 1 1],'HorizontalAlignment','center','Position',[0
introSize(4)-50 introSize(3) 25],'String','Welcome to the LTP','FontName','Times New
Roman','FontSize',18,'Visible','on','Style','text');
0033 uicontrol('Parent',introFig,'BackgroundColor',[1 1 1],'HorizontalAlignment','center','Position',[0
introSize(4)-75 introSize(3) 25],'String','Data Analysis Toolbox','FontName','Times New
Roman','FontSize',18,'Visible','on','Style','text');
0034
0035 loading = uicontrol('Parent',introFig,'BackgroundColor',[1 1
1],'HorizontalAlignment','left','Position',[(introSize(3)-50)/2 15 70 25],'String','loading','FontName','Times
New Roman','FontSize',10,'fontAngle','italic','Visible','on','Style','text','Tag','loading');
0036
0037 gif_image = 'loading.gif';
0038 delay_length = 0.1;
0039 gifSize = [23 23];
0040 axes('Parent',introFig,'Units','pixels','Position',[(introSize(3)-gifSize(1))/2 50 gifSize]);
0041 [handles.im,map] = imread(gif_image,'frames','all');
```

```
0042     handles.len = size(handles.im,4);
0043     handles.h1 = image(handles.im(:,:,1));
0044     colormap(map);
0045     axis normal;
0046     axis off;
0047     handles.guifig = gcf;
0048     handles.count = 1;
0049     handles.tmr = timer('TimerFcn',
{@TmrFcn,handles.guifig},'BusyMode','Queue','ExecutionMode','FixedRate','Period',delay_length);
0050     guidata(handles.guifig, handles);
0051     start(handles.tmr); %starts Timer
0052     guidata(handles.guifig, handles);
0053
0054     set(handles,'DeleteFcn',{@DeleteFunction,handles.guifig});
0055
0056
0057     function TmrFcn(src,event,handles)
0058         handles = guidata(handles);
0059         set(handles.h1,'CData',handles.im(:,:,handles.count)); %update the frame in the axis
0060         handles.count = handles.count + 1; %increment to next frame
0061
0062         if handles.count > handles.len %if the last frame is achieved intialise to first frame
0063             handles.count = 1;
0064         end
0065         guidata(handles.guifig, handles);
0066
0067         loadingStr = get(findobj('Tag','loading'),'String');
0068         switch loadingStr
0069             case 'loading....' loadingStr = 'loading';
0070             otherwise loadingStr = [loadingStr, '.'];
0071         end
0072
0073         if timePeriod == 2, set(findobj('Tag','loading'),'String',loadingStr); timePeriod = 0;
0074         else timePeriod = timePeriod +1;
0075         end
0076     end
0077
0078
0079     function DeleteFunction(src,event,handles)
0080         handles = guidata(handles);
0081         clear global timePeriod
0082         try stop(handles.tmr);delete(handles.tmr); catch end
0083     end
0084
0085     end
0086
0087
0088
```

Contents

PURPOSE ^

%% path: m/gui/ltpdaRepoGUI
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/ltpdaRepoGUI %%%%%%%%%%%%%%  
    <a href="matlab:help m/gui/ltpdaRepoGUI/createTable">m/gui/ltpdaRepoGUI/createTable</a> -  
- create nice-looking table based on javax.swing.JTable  
    <a href="matlab:help m/gui/ltpdaRepoGUI/muitable">m/gui/ltpdaRepoGUI/muitable</a> -  
creates a two dimensional graphicuitable component in a figure window.  
    <a href="matlab:help m/gui/ltpdaRepoGUI/repogui">m/gui/ltpdaRepoGUI/repogui</a> - a  
GUI to interact with an LTPDA Repository.  
    <a href="matlab:help m/gui/ltpdaRepoGUI/sqlResultsGUI">m/gui/ltpdaRepoGUI/sqlResultsGUI</a>  
- shows the results of a query in a new window.  
    <a href="matlab:help m/gui/ltpdaRepoGUI/uitabpanel">m/gui/ltpdaRepoGUI/uitabpanel</a> -  
adds a group of tabbed uipanel container objects to a figure.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [createTable.m](#)

createTable

PURPOSE [^]

createTable - create nice-looking table based on javax.swing.JTable

SYNOPSIS [^]

function [mtable, buttons] = createTable(pnContainer,headers,data,buttonsFlag,varargin)

DESCRIPTION [^]

```
createTable - create nice-looking table based on javax.swing.JTable

Syntax:
[mtable, buttons] = createTable (pnContainer, headers, data, buttonsFlag,
'PropName',PropValue, ...)

Input Parameters:
pnContainer - optional handle to container uipanel or figure. If empty/unsupplied then
current figure will be used
headers      - optional cell array of column header strings. If unsupplied then =
{'A','B','C'}
data        - optional vector/matrix (either scalar or cell array) of data values
buttonsFlag - optional flag indicating whether to create the table-manipulation buttons.
Default = true
'PropName',PropValue -
    optional list of property pairs (e.g.,
'AutoResizeMode',4,'Editable',false,'Position',[.1,.1,.5,.5])
    Note: PropName must be either an mtable property
('Visible','Editable','Position','Units',
'DataChangedCallback',...) or otherwise a Javax.swing.JTable property
('ShowGrid','Name',...). Abbreviated PropNames are unsupported for mtable properties (which
are few) - only for JTable

Output parameters:
mtable      - handle to mtable object (a Matlab object)
buttons     - handles to table manipulation buttons: [<appendRow> <insertRow> <deleteRow>
<deleteAll> <printAll>]

Examples:
[mtable, buttons] = createTable;
[mtable, buttons] = createTable(gcf,'column name');
mtable = createTable([],{'a','b','c','d'},{false,1.3,uint16(45),'ert'; true,pi,uint16(-
4),'defrgt'})
mtable =
createTable([],{'a','b','c','d'},magic(4),false,'AutoResizeMode',javax.swing.JTable.AUTO_RESIZE_
mtable =
createTable([],{'rads','sin','cos'},[pi,sin(pi),cos(pi)],'SelectionMode',javax.swing.ListSelectio

Usage:
The table automatically resizes to fill the pnContainer (you may modify this via the
'Position' property).
The table automatically sets the columns' cell editor and renderer based on the supplied
data. Logical values are
given a checkbox, strings are left-aligned (numbers are right-aligned). You can
always override the defaults.
You can change column widths by dragging the column borders on the header row.
You can sort columns by clicking the column header (once to sort descending, once again
to sort ascending and once
more for the unsorted view). Sorting multiple columns is done by control-clicking all
relevant columns (the
sorting icon is decreased in size for each additional minor sort col).
You can copy/paste any consecutive region of table cells, just as in Excel. You can
select entire rows or columns
by right-clicking their header. You can also paste Excel data directly, with Ctrl-
Shift-V (or use the context
menu by right-clicking) at the target table cell.
For additional tips about how to set multiple aspects of the table, refer to:
http://java.sun.com/docs/books/tutorial/uiswing/components/table.html
```

```

Programming tips/cues/examples:
mtable = creatTable(...)
jtable = mtable.getTable;
mtable.setVisible(false);
mtable.setCheckBoxEditor(1); % Set first column to a checkbox (see Note 2 below)
cb = javax.swing.JComboBox({'First','Last'}); cb.setEditable(true); % prepare an
editable drop-down CellEditor
editor = javax.swing.DefaultCellEditor(cb);
jtable.getColumnModel.getColumn(1).setCellEditor(editor); % assign this editor to second
column (see Note 2)
jtable.getColumnModel.getColumn(0).setMaxWidth(20); % Limit width of first (checkbox)
column (see Note 2)
mtable.setEditable(0,false); % Disable editing first column (see note 2 below)
renderer = javax.swing.table.DefaultTableCellRenderer; % or: renderer =
jtable.getColumnModel.getColumn(1).getCellRenderer
renderer.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT); % useful for numbers
rendered as strings e.g.: jtable.setValueAt(sprintf('%.1f',pi,rowIdx,colIdx))
jtable.getColumnModel.getColumn(1).setCellRenderer(renderer); % right-align second
column (see note 2)
data = cell(mtable.getData); % a cell matrix (mtable.getData is a java.lang.Object[][]
object, using base-1 indexing)
data = mtable.getTableModel.getDataVector; % a java.util.Vector object ([[false, 1.3,
45, ert], [true, 3.14,...]])
jtable.setValueAt(value,rowIdx,colIdx); % 0-based Idx - see Note 2 below
jtable.getModel.addRow({true, pi, int16(45), 'test'}); % appends a row to the bottom of
the table
mtable.DataChangedCallback = []; % used to temporarily disable data-change callbacks
mtable.DataChangedCallback = @myDataChange_Callback; % myDataChange_Callback is a Matlab
function

```

```

% Sample dataChange_Callback function
function dataChange_Callback(mtable, eventdata)
    if ~ishandle(mtable), return; end
    % Prevent re-entry here if the callback is not thread-safe - see Note 3 below
    eventDetails = eventdata.getEvent;
    modifiedColIdx = eventDetails.getColumn;
    modifiedRowIdx = eventDetails.getFirstRow;
    if modifiedColIdx>=0 && modifiedRowIdx>=0
        data = mtable.getData;
        newValue = data(modifiedRowIdx+1,modifiedColIdx+1); % see Note 2 below
        switch modifiedColIdx
            case ...
        end
    end
end

```

Notes:

1. Some (very few) JTable features are inconsistent or unavailable in different java versions. Type `'<a href="matlab:version -java">version -java</a>'` at the command prompt to see your specific java version.
2. Note that java uses 0-based indexing, while Matlab is 1-based. The returned mtable parameter is a Matlab object (so use 1-base), while mtable.getXXX returns java objects (0-based). jtable above is an example of a java object.
3. Modifying mtable.DataChangedCallback within the callback doesn't work - you need to use some global flag/mutex
4. The <Print> button uses Excel to parse and print the table
5. Due to Matlab limitations (specifically, of uitable/UitablePeer) the table is created as a direct child of the container figure (although it is visually positioned within pnContainer)
6. To enable sorting functionality, the attached TableSorter.jar file must be located in the java classpath. See the Matlab documentation for `<a href="matlab:doc javaclasspath">javaclasspath</a>`. Note that using `javaaddpath(...)` to set the path has a nasty side-effect (at least since Matlab 7.2) of clearing all globals! An alternative is to place the pathname for TableSorter.jar in the `<a href="matlab:which classpath.txt">classpath.txt</a>` file

Known issues/limitations:

- Column alignment not preserved during Print
- Print fails if Excel unavailable (maybe directly print tab-separated text data)
- Unable to add/delete rows or to print via context menu (right-click)
- Table is created as a direct child of figure, not pnContainer (see Note 5 above)

Bugs and suggestions:

Please send to Yair Altman (altmany at gmail dot com)

See also:

uitable, java, javaclasspath

Release history:

- 1.0 2007-03-09: initial version
- 1.1 2007-03-22: fixed selected row# on deletion of bottom row, main comment, missing

```
option; added header tooltip
```

```
25-02-08 Adapted and imported into LTPDA  
M Hewitson
```

```
$Id: createTable.m,v 1.2 2008/03/01 11:46:46 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [sqlResultsGUI](#) SQLRESULTSGUI shows the results of a query in a new window.

SUBFUNCTIONS [^]

 [function buttons = createManipulationButtons\(pnContainer, mtable\)](#)
 [function btTableInsertRow_Callback\(hObject, eventdata, handles\)](#)
 [function btTableAppendRow_Callback\(hObject, eventdata, handles\)](#)
 [function btTableDeleteRow_Callback\(hObject, eventdata, handles\)](#)
 [function btTableDeleteAll_Callback\(hObject, eventdata, handles\)](#)
 [function btTablePrintAll_Callback\(hObject, eventdata, handles\)](#)
 [function colStr = n2a\(c\)](#)
 [function stopEditing\(jtable\)](#)
 [function setVisibility\(hObject, enableStr\)](#)

SOURCE CODE [^]

```
0001 function [mtable, buttons] = createTable(pnContainer, headers, data, buttonsFlag, varargin)
0002 % createTable - create nice-looking table based on javax.swing.JTable
0003 %
0004 % Syntax:
0005 %     [mtable, buttons] = createTable (pnContainer, headers, data, buttonsFlag,
0006 %     'PropName', PropValue, ...)
0007 %
0008 % Input Parameters:
0009 %     pnContainer - optional handle to container uipanel or figure. If empty/unsupplied
0010 %     then current figure will be used
0011 %     headers     - optional cell array of column header strings. If unsupplied then =
0012 %     {'A','B','C'}
0013 %     data        - optional vector/matrix (either scalar or cell array) of data values
0014 %     buttonsFlag - optional flag indicating whether to create the table-manipulation
0015 %     buttons. Default = true
0016 %     'PropName', PropValue -
0017 %     optional list of property pairs (e.g.,
0018 %     'AutoResizeMode',4, 'Editable', false, 'Position', [.1, .1, .5, .5])
0019 %     Note: PropName must be either an mtable property
0020 %     ('Visible', 'Editable', 'Position', 'Units',
0021 %     'DataChangedCallback', ...) or otherwise a javax.swing.JTable
0022 %     property ('ShowGrid', 'Name', ...).
0023 %     Abbreviated PropNames are unsupported for mtable properties
0024 %     (which are few) - only for JTable
0025 %
0026 % Output parameters:
0027 %     mtable      - handle to mtable object (a Matlab object)
0028 %     buttons     - handles to table manipulation buttons: [<appendRow> <insertRow>
0029 %     <deleteRow> <deleteAll> <printAll>]
0030 %
0031 % Examples:
0032 %     [mtable, buttons] = createTable;
0033 %     [mtable, buttons] = createTable(gcf, 'column name');
0034 %     mtable = createTable([], {'a','b','c','d'}, {false, 1.3, uint16(45), 'ert';
0035 %     true, pi, uint16(-4), 'defrgt'})
0036 %     mtable =
0037 %     createTable([], {'a','b','c','d'}, magic(4), false, 'AutoResizeMode', javax.swing.JTable.AUTO_RESIZE_
0038 %
0039 %     mtable =
0040 %     createTable([], {'rads', 'sin', 'cos'}, [pi, sin(pi), cos(pi)], 'SelectionMode', javax.swing.ListSelectio
```



```

0028 %
0029 % Usage:
0030 %     The table automatically resizes to fill the pnContainer (you may modify this via
0031 %     the 'Position' property).
0032 %     The table automatically sets the columns' cell editor and renderer based on the
0033 %     supplied data. Logical values are
0034 %     given a checkbox, strings are left-aligned (numbers are right-aligned). You can
0035 %     always override the defaults.
0036 %     You can change column widths by dragging the column borders on the header row.
0037 %     You can sort columns by clicking the column header (once to sort descending, once
0038 %     again to sort ascending and once
0039 %     more for the unsorted view). Sorting multiple columns is done by control-
0040 %     clicking all relevant columns (the
0041 %     sorting icon is decreased in size for each additional minor sort col).
0042 %     You can copy/paste any consecutive region of table cells, just as in Excel. You
0043 %     can select entire rows or columns
0044 %     by right-clicking their header. You can also paste Excel data directly, with
0045 %     Ctrl-Shift-V (or use the context
0046 %     menu by right-clicking) at the target table cell.
0047 %     For additional tips about how to set multiple aspects of the table, refer to:
0048 %     <a
0049 %     href="http://java.sun.com/docs/books/tutorial/uiswing/components/table.html">http://java.sun.com,
0050 %
0051 % Programming tips/cues/examples:
0052 %     mtable = createTable(...)
0053 %     jTable = mtable.getTable();
0054 %     mtable.setVisible(false);
0055 %     jTable.setCheckBoxEditor(1); % Set first column to a checkbox (see Note 2 below)
0056 %     cb = javax.swing.JComboBox({'First','Last'}); cb.setEditable(true); % prepare an
0057 %     editable drop-down CellEditor
0058 %     editor = javax.swing.DefaultCellEditor(cb);
0059 %     jTable.getColumnModel.getColumn(1).setCellEditor(editor); % assign this editor to
0060 %     second column (see Note 2)
0061 %     jTable.getColumnModel.getColumn(0).setMaxWidth(20); % Limit width of first
0062 %     (checkbox) column (see Note 2)
0063 %     mtable.setEditable(0,false); % Disable editing first column (see note 2 below)
0064 %     renderer = javax.swing.table.DefaultTableCellRenderer; % or: renderer =
0065 %     jTable.getColumnModel.getColumn(1).getCellRenderer
0066 %     renderer.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT); % useful for
0067 %     numbers rendered as strings e.g.: jTable.setValueAt(sprintf('%1f',pi,rowIdx,colIdx))
0068 %     jTable.getColumnModel.getColumn(1).setCellRenderer(renderer); % right-align second
0069 %     column (see note 2)
0070 %     data = cell(mtable.getData); % a cell matrix (mtable.getData is a
0071 %     java.lang.Object[][] object, using base-1 indexing)
0072 %     data = mtable.getTableModel.getDataVector; % a java.util.Vector object ([[false,
0073 %     1.3, 45, ert], [true, 3.14,...]])
0074 %     jTable.setValueAt(value,rowIdx,colIdx); % 0-based Idx - see Note 2 below
0075 %     jTable.getModel.addRow({true, pi, int16(45), 'test'}); % appends a row to the
0076 %     bottom of the table
0077 %     mtable.DataChangedCallback = []; % used to temporarily disable data-change
0078 %     callbacks
0079 %     mtable.DataChangedCallback = @myDataChange_Callback; % myDataChange_Callback is a
0080 %     Matlab function
0081 %
0082 %     % Sample dataChange_Callback function
0083 %     function dataChange_Callback(mtable, eventdata)
0084 %         if ~ishandle(mtable), return; end
0085 %         % Prevent re-entry here if the callback is not thread-safe - see Note 3
0086 %         below
0087 %         eventDetails = eventdata.getEvent;
0088 %         modifiedColIdx = eventDetails.getColumn;
0089 %         modifiedRowIdx = eventDetails.getFirstRow;
0090 %         if modifiedColIdx>=0 && modifiedRowIdx>=0
0091 %             data = mtable.getData;
0092 %             newValue = data(modifiedRowIdx+1,modifiedColIdx+1); % see Note 2 below
0093 %             switch modifiedColIdx
0094 %                 case ...
0095 %                     end
0096 %             end
0097 %         end
0098 %     end
0099 %
0100 % Notes:
0101 %     1. Some (very few) JTable features are inconsistent or unavailable in different
0102 %     java versions. Type
0103 %     ' <a href="matlab:version -java">version -java</a>' at the command prompt to see
0104 %     your specific java version.
0105 %     2. Note that java uses 0-based indexing, while Matlab is 1-based. The returned
0106 %     jTable parameter is a Matlab object
0107 %     (so use 1-base), while mtable.getXXX returns java objects (0-based). jTable
0108 %     above is an example of a java object.
0109 %     3. Modifying mtable.DataChangedCallback within the callback doesn't work - you need
0110 %     to use some global flag/mutex
0111 %     4. The <Print> button uses Excel to parse and print the table
0112 %     5. Due to Matlab limitations (specifically, of uitable/UitablePeer) the table is
0113 %     created as a direct child of
0114 %     the container figure (although it is visually positioned within pnContainer)

```

```

00087 % 6. To enable sorting functionality, the attached TableSorter.jar file must be
located in the java classpath.
00088 % See the Matlab documentation for <a href="matlab:doc
javaclasspath">javaclasspath</a>. Note that using
00089 % javaaddpath(...) to set the path has a nasty side-effect (at least since Matlab
7.2) of clearing all globals!
00090 % An alternative is to place the pathname for TableSorter.jar in the <a
href="matlab:which classpath.txt">classpath.txt</a> file
00091 %
00092 % Known issues/limitations:
00093 % - Column alignment not preserved during Print
00094 % - Print fails if Excel unavailable (maybe directly print tab-separated text data)
00095 % - Unable to add/delete rows or to print via context menu (right-click)
00096 % - Table is created as a direct child of figure, not pnContainer (see Note 5 above)
00097 %
00098 % Bugs and suggestions:
00099 % Please send to Yair Altman (altmany at gmail dot com)
0100 %
0101 % See also:
0102 % uitable, java, javaclasspath
0103 %
0104 % Release history:
0105 % 1.0 2007-03-09: initial version
0106 % 1.1 2007-03-22: fixed selected row# on deletion of bottom row, main comment,
missing option; added header tooltip
0107 %
0108 %
0109 % 25-02-08 Adapted and imported into LTPDA
0110 % M Hewitson
0111 %
0112 % $Id: createTable.m,v 1.2 2008/03/01 11:46:46 hewitson Exp $
0113 %
0114 %
0115 %
0116 % License to use and modify this code is granted freely to all interested, as long as
the original author is
0117 % referenced and attributed as such. The original author maintains the right to be
solely associated with this work.
0118 %
0119 % Programmed and Copyright by Yair M. Altman: altmany(at)gmail.com
0120 % $Revision: 1.2 $ $Date: 2008/03/01 11:46:46 $
0121 %
0122 %try
0123 % Ensure that java swing is enabled...
0124 if ~usejava('swing')
0125     error('createTable:NeedSwing','Java tables require Java Swing.');
```

```

0126 end
0127
0128 % Create a panel spanning entire figure area, if panel handle was not supplied
0129 if (nargin < 1) || isempty(pnContainer) || ~ishandle(pnContainer)
0130     pnContainer = uipanel('parent',gcf,'tag','TablePanel');
```

```

0131 end
0132 pnContainerPos = getpixelposition(pnContainer,1);
0133 if isa(handle(pnContainer),'figure')
0134     pnContainerPos(1:2) = 0;
0135 end
0136
0137 % Get handle to parent figure
0138 hFig = ancestor(pnContainer,'figure');
```

```

0139
0140 % Determine whether table manipulation buttons are requested
0141 if nargin < 4 || isempty(buttonsFlag) || ~(isnumeric(buttonsFlag) ||
islogical(buttonsFlag))
0142     if nargin >= 4, varargin = {buttonsFlag, varargin{:}}; end
0143     buttonsFlag = true;
0144 end
0145 if buttonsFlag
0146     margins = [1,30,0,-30]; % With buttons
0147 else
0148     margins = [1,1,0,0]; % No buttons
0149 end
0150
0151 % Get the uitable's required position within pnContainer
0152 tablePosition = pnContainerPos + margins; % Relative to the figure
0153
0154 % Set default header names, if not supplied
0155 if nargin < 2
0156     headers = {'A','B','C'}; % 3 columns by default
0157 elseif isempty(headers)
0158     headers = cell(1,size(data,2));
0159 elseif ischar(headers)
0160     headers = {headers};
0161 end
0162
0163 % Start with dummy data, just so that uitable can be initialized (or use supplied
data, if available)

```

```

0164     if nargin < 3 || isempty(data)
0165         numRows = 0;
0166         numCols = length(headers);
0167         data = zeros(1,numCols);
0168     else
0169         numRows = size(data,1);
0170         numCols = size(data,2);
0171     end
0172     % Convert to cell-format (if not so already)
0173     if ~iscell(data)
0174         data = mat2cell(data,ones(1,size(data,1)),ones(1,numCols));
0175     end
0176
0177     % Create a sortable uitable within pnHandle
0178     mtable = uitable(hFig, 'position',tablePosition, 'Data',data,
'ColumnNames',headers);
0179     mtable.setNumRows(numRows);
0180     set(mtable,'units','normalized'); % this will resize the table whenever its
container is resized
0181
0182     % jTable is the underlying java JTable - access to lots more functionality...
0183     % Note: actually, jTable is a
com.mathworks.hg.peer.UitablePeer$PeerSpreadsheetTable object, but this extends
0184     % ^^^^ javax.swing.JTable, so for all practical purposes you may use it as a
JTable
0185     jTable = mtable.getTable;
0186
0187     % Fix for JTable focus bug : see
http://bugs.sun.com/bugdatabase/view_bug.do;WuuT?bug_id=4709394
0188     % Taken from: http://xtargets.com/snippets/posts/show/37
0189     jTable.putClientProperty('terminateEditOnFocusLost', java.lang.Boolean.TRUE);
0190
0191     % We want to use sorter, not data model...
0192     % unfortunately, UitablePeer expects DefaultTableModel (not TableSorter) so we
need a modified UitablePeer class
0193     % however, UitablePeer is a Matlab class, so instead let's use a modified
TableSorter and attach it to the Model
0194     %sorter = com.mathworks.toolbox.dasstudio.util.TableSorter; % Failed attempt...
0195     %sorter = com.mathworks.mmswing.DefaultSortableTable; % ...another failed
attempt...
0196     if ~isempty(which('TableSorter'))
0197         % Add TableSorter as TableModel listener
0198         sorter = TableSorter(jTable.getModel()); %(table.getTableModel);
0199         %tablePeer = UitablePeer(sorter); % This is not accepted by UitablePeer... -
see comment above
0200         jTable.setModel(sorter);
0201         sorter.setTableHeader(jTable.getTableHeader());
0202
0203         % Set the header tooltip (with sorting instructions)
0204         jTable.getTableHeader().setToolTipText('<html>&nbsp;<b>Click</b> to sort up;
<b>Shift-click</b> to sort down<br>&nbsp;<b>Ctrl-click</b> (or <b>Ctrl-Shift-click</b>) to
sort secondary&nbsp;<br>&nbsp;<b>Click again</b> to change sort direction<br>&nbsp;<b>Click a
third time</b> to return to unsorted view<br>&nbsp;<b>Right-click</b> to select entire
column</html>');
0205     else
0206         % Set the header tooltip (no sorting instructions...)
0207         jTable.getTableHeader().setToolTipText('<html>&nbsp;<b>Click</b> to select
entire column<br>&nbsp;<b>Ctrl-click</b> (or <b>Shift-click</b>) to select multiple
columns&nbsp;</html>');
0208     end
0209
0210     % Store the uitable's handle within the pnContainer's userdata, for later use
0211     set(pnContainer,'userdata',[get(pnContainer,'userdata'), mtable]); % add to
parent userdata, so we have a handle for deletion
0212
0213     % Enable multiple row selection, auto-column resize, and auto-scrollbars
0214     scroll = mtable.TableScrollPane;
0215     scroll.setVerticalScrollBarPolicy(scroll.VERTICAL_SCROLLBAR_AS_NEEDED);
0216     scroll.setHorizontalScrollBarPolicy(scroll.HORIZONTAL_SCROLLBAR_AS_NEEDED);
0217
jTable.setSelectionMode(javax.swing.ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
0218     jTable.setAutoResizeMode(jTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS)
0219
0220     % Set the jTable name based on the containing panel's tag
0221     basicTagName = get(pnContainer,'tag');
0222     jTable.setName([basicTagName 'Table']);
0223
0224     % Move the selection to first table cell (if any data available)
0225     if (jTable.getRowCount > 0)
0226         jTable.changeSelection(0,0,false,false);
0227     end
0228
0229     % Process optional args
0230     for argIdx = 1 : 2 : length(varargin)
0231         if argIdx<2
0232             % We need this pause to let java complete all table rendering

```

```

0233         % TODO: We should really use calls to awtinvoke() instead, though...
0234         pause(0.05);
0235     end
0236     if (length(varargin) > argIdx) % ensure the arg value is there...
0237         varargin{argIdx}(1) = upper(varargin{argIdx}(1)); % property names always
start with capital letters...
0238         propMethodName = ['set' varargin{argIdx}];
0239         if ismethod(mtable,propMethodName)
0240             set(mtable,varargin{argIdx},varargin{argIdx+1});
0241         else
0242             %javaMethod(propMethodName, jtable, varargin{argIdx+1});
0243             set(jtable,varargin{argIdx},varargin{argIdx+1});
0244         end
0245     end
0246 end % for argIdx
0247
0248 % Create table manipulation buttons
0249 if buttonsFlag
0250     buttons = createManipulationButtons(pnContainer,mtable);
0251 else
0252     buttons = [];
0253 end
0254 %catch
0255 % Insert your code here
0256 %handleError;
0257 %end
0258
0259
0260 %% --- Executes on button press in btInsert.
0261 function buttons = createManipulationButtons(pnContainer, mtable)
0262 % pnContainer handle to container uipanel
0263 % mtable handle to mtable (Matlab) object
0264 %try
0265     btAppendRow = uicontrol('tag','btTableAppendRow',
'callback',@btTableAppendRow_Callback,'position',[10,5,60,20], 'string','Append',
'parent',pnContainer, 'userdata',mtable);
0266     btInsertRow = uicontrol('tag','btTableInsertRow',
'callback',@btTableInsertRow_Callback,'position',[80,5,60,20], 'string','Insert',
'parent',pnContainer, 'userdata',mtable);
0267     btDeleteRow = uicontrol('tag','btTableDeleteRow',
'callback',@btTableDeleteRow_Callback,'position',[150,5,60,20], 'string','Delete',
'parent',pnContainer, 'userdata',mtable);
0268     btDeleteAll = uicontrol('tag','btTableDeleteAll',
'callback',@btTableDeleteAll_Callback,'position',[220,5,60,20], 'string','Delete All',
'parent',pnContainer, 'userdata',mtable);
0269     btPrintAll = uicontrol('tag','btTablePrintAll',
'callback',@btTablePrintAll_Callback,'position',[290,5,60,20], 'string','Print',
'parent',pnContainer, 'userdata',mtable);
0270     buttons = [btInsertRow btAppendRow btDeleteRow btDeleteAll btPrintAll];
0271     if mtable.getNumRows < 1
0272         setVisibility(pnContainer, 'off');
0273     end
0274 %catch
0275 % Insert your code here
0276 %handleError;
0277 %end
0278
0279
0280 %% --- Executes on button press in btInsert.
0281 % Insert a new row immediately before the current row
0282 function btTableInsertRow_Callback(hObject, eventdata, handles) %#ok
0283 % hObject handle to btTableInsertRow (see GCBO)
0284 % eventdata reserved - to be defined in a future version of MATLAB
0285 % handles structure with handles and user data (see GUIDATA)
0286 %try
0287     mtable = get(hObject,'userdata');
0288     jtable = mtable.getTable;
0289
0290 % Stop any current editing
0291 stopEditing(jtable);
0292
0293 % Insert the new row immediately before the current row
0294 newRowData = cell(1,mtable.getNumColumns); % empty data
0295 mtable.getTableModel.insertRow(max(0,jtable.getSelectedRow), newRowData);
0296 %catch
0297 % Insert your code here
0298 %handleError;
0299 %end
0300
0301
0302 %% --- Executes on button press in btAppend.
0303 % Insert a new row as the last row in the table
0304 function btTableAppendRow_Callback(hObject, eventdata, handles) %#ok
0305 % hObject handle to btTableAppendRow (see GCBO)
0306 % eventdata reserved - to be defined in a future version of MATLAB
0307 % handles structure with handles and user data (see GUIDATA)

```



```

0308 %try
0309 mtable = get(hObject,'userdata');
0310 jtable = mtable.getTable;
0311
0312 % Stop any current editing
0313 stopEditing(jtable);
0314
0315 % Add a new row at the bottom of the data table
0316 newRowData = cell(1,mtable.getNumColumns); % empty data
0317 mtable.getTableModel.addRow(newRowData);
0318
0319 % Move the selection to Column A of this new row
0320 jtable.changeSelection(jtable.getRowCount-1,0,false,false);
0321
0322 % There must be at least one table row now, so display the table in any case
0323 mtable.setVisible(true);
0324 setVisibility(hObject, 'on');
0325 %catch
0326 % Insert your code here
0327 %handleError;
0328 %end
0329
0330
0331 %% --- Executes on button press in btDelete.
0332 % If there are any rows displayed, then delete the currently-selected row
0333 function btTableDeleteRow_Callback(hObject, eventdata, handles) %#ok
0334 % hObject handle to btTableDeleteRow (see GCBO)
0335 % eventdata reserved - to be defined in a future version of MATLAB
0336 % handles structure with handles and user data (see GUIDATA)
0337 %try
0338 mtable = get(hObject,'userdata');
0339 jtable = mtable.getTable;
0340
0341 % Stop any current editing
0342 stopEditing(jtable);
0343
0344 % If there are any rows displayed, then delete the currently-selected row
0345 rowCount = jtable.getRowCount;
0346 if (rowCount > 0) % might be==0 during slow processing & user double-click
0347     currentRow = max(0,jtable.getSelectedRow);
0348     currentCol = max(0,jtable.getSelectedColumn);
0349     mtable.getTableModel.removeRow(currentRow);
0350     if currentRow >= rowCount-1
0351         jtable.changeSelection(currentRow-1, currentCol, false, false);
0352     end
0353 end
0354 if (jtable.getRowCount <= 0)
0355     %table.setVisible(false);
0356     setVisibility(hObject, 'off');
0357 end
0358 %catch
0359 % Insert your code here
0360 %handleError;
0361 %end
0362
0363
0364 %% --- Executes on button press in btDeleteAll.
0365 % Deletes all table rows
0366 function btTableDeleteAll_Callback(hObject, eventdata, handles) %#ok
0367 % hObject handle to btTableDeleteAll (see GCBO)
0368 % eventdata reserved - to be defined in a future version of MATLAB
0369 % handles structure with handles and user data (see GUIDATA)
0370 %try
0371 mtable = get(hObject,'userdata');
0372 jtable = mtable.getTable;
0373
0374 % Stop any current editing
0375 stopEditing(jtable);
0376
0377 % Delete all table rows
0378 mtable.setNumRows(0);
0379
0380 % Hide irrelevant controls
0381 %mtable.setVisible(false);
0382 setVisibility(hObject, 'off');
0383 %catch
0384 % Insert your code here
0385 %handleError;
0386 %end
0387
0388
0389 %% --- Executes on button press in btPrint.
0390 % Prints the table via Excel
0391 function btTablePrintAll_Callback(hObject, eventdata, handles) %#ok
0392 % hObject handle to btTablePrintAll (see GCBO)
0393 % eventdata reserved - to be defined in a future version of MATLAB

```

```

0394 % handles structure with handles and user data (see GUIDATA)
0395 persistent hExcel
0396 try
0397     mtable = get(hObject, 'userdata');
0398
0399     % Try to open an Excel COM server
0400     % Note: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odc_2003_ta/html/odc_landoffice03_vba.asp
0401     try
0402         % try to reuse an existing (pre-opened) COM server
0403         % If we can't access the ActiveX parent, it means it's closed
0404         parent = hExcel.parent; %#ok
0405     catch
0406         hExcel = actxserver('excel.application');
0407     end
0408
0409     % Try to open the requested document
0410     hExcel.Workbooks.Add;
0411
0412     % Format field headers
0413     headers = cell(mtable.getColumnNames)';
0414     if ~isempty(headers)
0415         hExcel.Range(['A1:' n2a(length(headers)) '1']).Select;
0416         hExcel.Selection.Value = headers;
0417         hExcel.Selection.Font.Bold = true; % bold
0418         hExcel.Selection.Font.Color = hex2dec('0000FF'); % Red
0419         hExcel.Selection.Border.Item(4).Weight = 3; % underline
0420         hExcel.Selection.Cells.HorizontalAlignment = -4108; % =xlCenter
0421     end
0422
0423     % Set the data from the table
0424     data = cell(mtable.data);
0425     if ~isempty(headers)
0426         hExcel.Range(['A2:' n2a(size(data,2)) num2str(1+size(data,1))]).Select;
0427         hExcel.Selection.Value = data;
0428     end
0429
0430     % TODO: Change checkbox fields to boolean (TRUE/empty)
0431
0432     % Other formats
0433     %hExcel.Cells.HorizontalAlignment = -4108; % =xlCenter % TODO: preserve original
jtable column alignment
0434     hExcel.Cells.EntireColumn.AutoFit;
0435     hExcel.ActiveSheet.DisplayRightToLeft = false;
0436     set(hExcel.ActiveSheet.PageSetup, 'LeftMargin', hExcel.InchesToPoints(0.1), ...
0437         'RightMargin', hExcel.InchesToPoints(0.1), ...
0438         'HeaderMargin', hExcel.InchesToPoints(0), ...
0439         'FooterMargin', hExcel.InchesToPoints(0.1), ...
0440         'TopMargin', 120, ...
0441         'FitToPagesWide', 1, ...
0442         'FitToPagesTall', 1, ...
0443         'Orientation', 'xlPortrait', ...
0444         'LeftHeader', '&D &T', ...
0445         'CenterHeader', char(mtable.getTable.getName),
0446         'RightHeader', '&G');
0447
0448     % Send to printer
0449     hExcel.ActiveWindow.SelectedSheets.PrintOut;
0450
0451     % Close the workbook
0452     invoke(hExcel.ActiveWindow, 'close', false);
0453 catch
0454     % Insert your code here
0455     %handleError;
0456     err = lasterror;
0457     try invoke(hExcel.ActiveWindow, 'close', false); catch end; % just in case of a
printing error
0458     rethrow(err);
0459 end
0460
0461 %% --- Convert col # format to 'A','B','C','AA',... format
0462 % Thanks Brett Shoelson, via CSSM
0463 function colStr = n2a(c)
0464     t = [floor((c-1)/26)+64, rem(c-1,26)+65];
0465     if (t(1)<65), t(1) = []; end
0466     colStr = char(t);
0467
0468
0469 %% --- Executes on button press in btInsert.
0470 function stopEditing(jtable)
0471     %try
0472     component = jtable.getEditorComponent;
0473     if ~isempty(component)
0474         event = javax.swing.event.ChangeEvent(component);
0475

```

```
0476         jtable.editingStopped(event);
0477     end
0478 %catch
0479 % Insert your code here
0480 %handleError;
0481 %end
0482
0483
0484 %% --- Utility function to set visibility of row manipulation buttons
0485 function setVisible(hObject, enableStr)
0486 % hObject    handle to some element within the figure
0487 % enableStr  'on' or 'off'
0488 %try
0489     hParent = ancestor(hObject, 'figure');
0490     set(findall(hParent, 'tag', 'btTableInsertRow'), 'enable', enableStr);
0491     set(findall(hParent, 'tag', 'btTableDeleteRow'), 'enable', enableStr);
0492     set(findall(hParent, 'tag', 'btTableDeleteAll'), 'enable', enableStr);
0493     set(findall(hParent, 'tag', 'btTablePrintAll'), 'enable', enableStr);
0494 %catch
0495 % Insert your code here
0496 %handleError;
0497 %end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

mutable

PURPOSE ^

UITABLE creates a two dimensional graphic uitable component in a figure window.

SYNOPSIS ^

function [table, container] = mutable(varargin)

DESCRIPTION ^

UITABLE creates a two dimensional graphic uitable component in a figure window. UITABLE creates a 1x1 uitable object using default property values in a figure window.

UITABLE(numrows,numcolumns) creates a uitable object with specified number of rows and columns.

UITABLE(data,columnNames) creates a uitable object with the specified data and columnNames. Data can be a cell array or a vector and columnNames should be cell arrays.

UITABLE('PropertyName1',value1,'PropertyName2',value2,...) creates a uitable object with specified property values. MATLAB uses default property values for any property not explicitly set. The properties that user can set are: ColumnNames, Data, GridColor, NumColumns, NumRows, Position, ColumnWidth and RowHeight.

UITABLE(figurehandle,...) creates a uitable object in the figure window specified by the figure handle.

HANDLE = UITABLE(...) creates a uitable object and returns its handle.

Properties:

ColumnNames: Cell array of strings for column names.
Data: Cell array of values to be displayed in the table.
GridColor: string, RGB vector.
NumColumns: int specifying number of columns.
NumRows: int specifying number of rows.
Parent: Handle to figure or uipanel. If not specified, it is(gcf).
Position: 4 element vector specifying the position.
ColumnWidth: int specifying the width of columns.
RowHeight: int specifying the height of columns.

Enabled: Boolean specifying if a column is enabled.
Editable: Boolean specifying if a column is editable.
Units: String - pixels/normalized/inches/points/centimeters.
Visible: Boolean specifying if table is visible.
DataChangedCallback - Callback function name or handle.

Examples:

```
t = uitable(3, 2);
```

Creates a 3x2 empty uitable object in a figure window.

```
f = figure;  
t = uitable(f, rand(5), {'A', 'B', 'C', 'D', 'E'});
```

Creates a 5x5 uitable object in a figure window with the specified data and the column names.

```
data = rand(3);  
colnames = {'X-Data', 'Y-Data', 'Z-Data'};  
t = uitable(data, colnames,'Position',[20 20 250 100]);
```

Creates a uitable object with the specified data and column names and the specified Position.

See also AWTCREATE, AWTINVOKE, JAVACOMPONENT, UITREE, UITREENODE

Adapted for LTPDA by M Hewitson
- a direct copy of MATLAB's uitable.m

\$Id: muitable.m,v 1.3 2008/02/26 08:54:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

 [function componentDelete\(src, evd\)](#)

SOURCE CODE [^]

```

0001 function [table, container] = muitable(varargin)
0002
0003 % Uitable creates a two dimensional graphic uitable component in a figure window.
0004 %     Uitable creates a 1x1 uitable object using default property values in
0005 %     a figure window.
0006 %
0007 %     Uitable(numrows,numcolumns) creates a uitable object with specified
0008 %     number of rows and columns.
0009 %
0010 %     Uitable(data,columnNames) creates a uitable object with the specified
0011 %     data and columnNames. Data can be a cell array or a vector and
0012 %     columnNames should be cell arrays.
0013 %
0014 %     Uitable('PropertyName1',value1,'PropertyName2',value2,...) creates a
0015 %     uitable object with specified property values. MATLAB uses default
0016 %     property values for any property not explicitly set. The properties
0017 %     that user can set are: ColumnNames, Data, GridColor, NumColumns,
0018 %     NumRows, Position, ColumnWidth and RowHeight.
0019 %
0020 %     Uitable(figurehandle,...) creates a uitable object in the figure
0021 %     window specified by the figure handle.
0022 %
0023 %     HANDLE = Uitable(...) creates a uitable object and returns its handle.
0024 %
0025 %     Properties:
0026 %
0027 %     ColumnNames: Cell array of strings for column names.
0028 %     Data:        Cell array of values to be displayed in the table.
0029 %     GridColor:   string, RGB vector.
0030 %     NumColumns:  int specifying number of columns.
0031 %     NumRows:     int specifying number of rows.
0032 %     Parent:      Handle to figure or uipanel. If not specified, it is gcf.
0033 %     Position:    4 element vector specifying the position.
0034 %     ColumnWidth: int specifying the width of columns.
0035 %     RowHeight:   int specifying the height of columns.
0036 %
0037 %     Enabled:     Boolean specifying if a column is enabled.
0038 %     Editable:    Boolean specifying if a column is editable.
0039 %     Units:       String - pixels/normalized/inches/points/centimeters.
0040 %     Visible:     Boolean specifying if table is visible.
0041 %     DataChangedCallback - Callback function name or handle.
0042 %
0043 %
0044 %     Examples:
0045 %
0046 %     t = uitable(3, 2);
0047 %
0048 %     Creates a 3x2 empty uitable object in a figure window.
0049 %
0050 %     f = figure;
0051 %     t = uitable(f, rand(5), {'A', 'B', 'C', 'D', 'E'});
0052 %
0053 %     Creates a 5x5 uitable object in a figure window with the specified
0054 %     data and the column names.
0055 %
0056 %     data = rand(3);
0057 %     colnames = {'X-Data', 'Y-Data', 'Z-Data'};
0058 %     t = uitable(data, colnames,'Position',[20 20 250 100]);
0059 %
0060 %     Creates a uitable object with the specified data and column names and

```

```

0061 % the specified Position.
0062 %
0063 % See also AWTCREATE, AWTINVOKE, JAVACOMPONENT, UITREE, UITREENODE
0064 %
0065 % Adapted for LTPDA by M Hewitson
0066 % - a direct copy of MATLAB's uitable.m
0067 %
0068 % $Id: mutable.m,v 1.3 2008/02/26 08:54:43 hewitson Exp $
0069 %
0070
0071
0072 % Copyright 2002-2006 The MathWorks, Inc.
0073 % $Revision: 1.3 $ $Date: 2008/02/26 08:54:43 $
0074
0075 % Release: R14. This feature will not work in previous versions of MATLAB.
0076
0077 % Setup and P-V parsing
0078
0079 error(javachk('awt'));
0080 error(nargoutchk(0,2,nargout));
0081
0082 parent = [];
0083 numargs = nargin;
0084
0085 datastatus=false; columnstatus=false;
0086 rownum = 1; colnum = 1; % Default to a 1x1 table.
0087 position = [20 20 200 200];
0088 combo_box_found = false;
0089 check_box_found = false;
0090
0091 import com.mathworks.hg.peer.UitablePeer;
0092
0093 if (numargs > 0 && isscalar(varargin{1}) && ishandle(varargin{1}) && ...
0094     isa(handle(varargin{1}), 'figure'))
0095     parent = varargin{1};
0096     varargin = varargin(2:end);
0097     numargs = numargs - 1;
0098 end
0099
0100 if (numargs > 0 && isscalar(varargin{1}) && ishandle(varargin{1}))
0101     if ~isa(varargin{1}, 'javax.swing.table.DefaultTableModel')
0102         error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter: ',
varargin{1}]);
0103     end
0104     data_model = varargin{1};
0105     varargin = varargin(2:end);
0106     numargs = numargs - 1;
0107
0108 elseif ((numargs > 1) && isnumeric(varargin{1}) && isnumeric(varargin{2}))
0109     if (isnumeric(varargin{1}) && isnumeric(varargin{2}))
0110         rownum = varargin{1};
0111         colnum = varargin{2};
0112
0113         varargin = varargin(3:end);
0114         numargs = numargs-2;
0115     else
0116         error('MATLAB:uitable:InputMustBeScalar', 'When using Uitable numRows and
numcols have to be numeric scalars. ');
0117     end
0118
0119 elseif ((numargs > 1) && isequal(size(varargin{2},1), 1) && iscell(varargin{2}))
0120     if (size(varargin{1},2) == size(varargin{2},2))
0121         if (isnumeric(varargin{1}))
0122             varargin{1} = num2cell(varargin{1});
0123         end
0124     else
0125         error('MATLAB:uitable:MustMatchInfo', 'Number of column names must match number
of columns in data ');
0126     end
0127     data = varargin{1}; datastatus = true;
0128     coln = varargin{1+1}; columnstatus = true;
0129
0130     varargin = varargin(3:end);
0131     numargs = numargs-2;
0132 end
0133
0134 for i = 1:2:numargs-1
0135     if (~ischar(varargin{i}))
0136         error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter: ',
varargin{i}]);
0137     end
0138     switch lower(varargin{i})
0139     case 'data'
0140         if (isnumeric(varargin{i+1}))
0141             varargin{i+1} = num2cell(varargin{i+1});
0142         end

```

```

0143         data      = varargin{i+1};
0144         datastatus = true;
0145
0146         case 'columnnames'
0147             if (iscell(varargin{i+1}))
0148                 coln      = varargin{i+1};
0149                 columnstatus = true;
0150             else
0151                 error('MATLAB:uitable:InvalidCellArray', 'When using Uitable Column data
should be 1xn cell array')
0152             end
0153
0154         case 'numrows'
0155             if (isnumeric(varargin{i+1}))
0156                 rownum = varargin{i+1};
0157             else
0158                 error('MATLAB:uitable:NumrowsMustBeScalar', 'numrows has to be a
scalar')
0159             end
0160
0161         case 'numcolumns'
0162             if (isnumeric(varargin{i+1}))
0163                 colnum = varargin{i+1};
0164             else
0165                 error('MATLAB:uitable:NumcolumnsMustBeScalar', 'numcolumns has to be a
scalar')
0166             end
0167
0168         case 'gridcolor'
0169             if (ischar(varargin{i+1}))
0170                 gridcolor = varargin{i+1};
0171             else if (isnumeric(varargin{i+1}) && (numel(varargin{i+1}) == 3))
0172                 gridcolor = varargin{i+1};
0173             else
0174                 error('MATLAB:uitable:InvalidString', 'gridcolor has to be a valid
string')
0175             end
0176
0177         case 'rowheight'
0178             if (isnumeric(varargin{i+1}))
0179                 rowheight = varargin{i+1};
0180             else
0181                 error('MATLAB:uitable:RowheightMustBeScalar', 'rowheight has to be a
scalar')
0182             end
0183
0184         case 'parent'
0185             if ishandle(varargin{i+1})
0186                 parent = varargin{i+1};
0187             else
0188                 error('MATLAB:uitable:InvalidParent', 'parent must be a valid handle')
0189             end
0190
0191         case 'position'
0192             if (isnumeric(varargin{i+1}))
0193                 position = varargin{i+1};
0194             else
0195                 error('MATLAB:uitable:InvalidPosition', 'position has to be a 1x4
numeric array')
0196             end
0197
0198         case 'columnwidth'
0199             if (isnumeric(varargin{i+1}))
0200                 columnwidth = varargin{i+1};
0201             else
0202                 error('MATLAB:uitable:ColumnwidthMustBeScalar', 'columnwidth has to be a
scalar')
0203             end
0204
0205         otherwise
0206             error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter: ',
varargin{i}]);
0207     end
0208 end
0209
0210 % ---combo/check box detection--- %
0211 if (datastatus)
0212     if (iscell(data))
0213         rownum = size(data,1);
0214         colnum = size(data,2);
0215         combo_count = 0;
0216         check_count = 0;
0217         combo_box_data = num2cell(zeros(1, colnum));
0218         combo_box_colnum = zeros(1, colnum);
0219         check_box_column = zeros(1, colnum);
0220         for j = 1:rownum

```

```

0221         for k = 1:colnum
0222             if (iscell(data{j,k}))
0223                 combo_box_found = true;
0224                 combo_count = combo_count + 1;
0225                 combo_box_data{combo_count} = data{j,k};
0226                 combo_box_column(combo_count) = k;
0227                 dc = data{j,k};
0228                 data{j,k} = dc{1};
0229             else
0230                 if(islogical(data{j,k}))
0231                     check_box_found = true;
0232                     check_count = check_count + 1;
0233                     check_box_column(check_count) = k;
0234                 end
0235             end
0236         end
0237     end
0238 end
0239 end
0240
0241 %%%%%%%%%%%%%%%
0242 % Check the validity of the parent and/or create a figure.
0243 if isempty(parent)
0244     parent =(gcf); % Get the current figure. Create one if not available
0245 end
0246
0247 if ( columnstatus && datastatus )
0248     if(size(data,2) ~= size(coln,2))
0249         error('MATLAB:NeedSameNumberColumns', 'Number of columns in both Data and
ColumnNames should match');
0250     end
0251 elseif ( ~columnstatus && datastatus )
0252     for i=1:size(data,2)
0253         coln{i} = num2str(i);
0254     end
0255     columnstatus = true;
0256 elseif ( columnstatus && ~datastatus)
0257     error('MATLAB:uitable:NoDataProvided', 'No Data provided along with ColumnNames');
0258 end
0259
0260 if (~exist('data_model', 'var'))
0261     data_model = javax.swing.table.DefaultTableModel;
0262 end
0263 if exist('rownum', 'var')
0264     data_model.setRowCount(rownum);
0265 end
0266 if exist('colnum', 'var')
0267     data_model.setColumnCount(colnum);
0268 end
0269
0270 table_h= UitablePeer(data_model);
0271
0272 % We should have valid data and column names here.
0273 if (datastatus), table_h.setData(data); end;
0274 if (columnstatus), table_h.setColumnNames(coln); end;
0275
0276 if (combo_box_found),
0277     for i=1:combo_count
0278         table_h.setComboBoxEditor(combo_box_data(i), combo_box_column(i));
0279     end
0280 end
0281 if (check_box_found),
0282     for i = 1: check_count
0283         table_h.setCheckBoxEditor(check_box_column(i));
0284     end
0285 end
0286
0287 % pass the specified parent and let javacomponent decide its validity.
0288 [obj, container] = javacomponent(table_h, position, parent);
0289 % javacomponent returns a UDD handle for the java component passed in.
0290 table = obj;
0291
0292 % Have to do a drawnow here to make the properties stick. Try to restrict
0293 % the drawnow call to only when it is absolutely required.
0294 flushed = false;
0295 if exist('gridcolor', 'var')
0296     pause(.1); drawnow;
0297     flushed = true;
0298     table_h.setGridColor(gridcolor);
0299 end
0300 if exist('rowheight', 'var')
0301     if (~flushed)
0302         drawnow;
0303     end
0304     table_h.setRowHeight(rowheight);
0305 end

```

```
0306 if exist('columnwidth', 'var')
0307     table_h.setColumnWidth(columnwidth);
0308 end;
0309
0310
0311
0312 % Add a predestroy listener so we can call cleanup on the table.
0313 % addlistener(table, 'ObjectBeingDestroyed', {@componentDelete});
0314
0315 function componentDelete(src, evd)                                %#ok
0316 % Clean up the table here so it disengages all its internal listeners.
0317 src.cleanup;
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > repogui.m

repogui

PURPOSE [^]

REPOGUI a GUI to interact with an LTPDA Repository.

SYNOPSIS [^]

function varargout = repogui(varargin)

DESCRIPTION [^]

```
REPOGUI a GUI to interact with an LTPDA Repository.
M Hewitson
$Id: repogui.m,v 1.16 2008/03/29 20:25:11 mauro Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [uitabpanel](#) UITABPANEL adds a group of tabbed uipanel container objects to a figure.

This function is called by:

SUBFUNCTIONS [^]

 [function panel = create_quicklook_panel\(panel, Gproperties\)](#)
 [function panel = create_retrieve_panel\(panel, Gproperties\)](#)
 [function panel = create_query_panel\(panel, Gproperties\)](#)
 [function panel = create_submit_panel\(panel, Gproperties\)](#)
 [function panel = create_connection_panel\(panel, Gproperties\)](#)

SOURCE CODE [^]

```
0001 function varargout = repogui(varargin)
0002
0003 % REPOGUI a GUI to interact with an LTPDA Repository.
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: repogui.m,v 1.16 2008/03/29 20:25:11 mauro Exp $
0008 %
0009
0010 % addpath('callbacks');
0011
0012 %% Check if I exist already
0013 id = findobj('Tag', 'LTPDARepomainfig');
0014 if ~isempty(id)
0015     figure(id)
0016     return
0017 end
0018
0019 %% Some initial setup
0020
0021 Gproperties.Gcol      = [240 240 240]/255;
0022 Gproperties.Gwidth    = 800;
0023 Gproperties.Gheight   = 650;
0024 Gproperties.Gborder   = 10;
0025 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0026
```



```

0027 Gproperties.Screen = get(0, 'screensize');
0028 Gproperties.Gposition = [Gproperties.Screen(3)/2-Gproperties.Gwidth/2 ...
0029     Gproperties.Screen(4)/2-Gproperties.Gheight/2 ...
0030     Gproperties.Gwidth ...
0031     Gproperties.Gheight];
0032
0033 % Initialize and hide the GUI as it is being constructed.
0034 mainfig = figure('Name', 'LTPDA Repository GUI', ...
0035     'NumberTitle', 'off', ...
0036     'Visible', 'off', ...
0037     'Position', Gproperties.Gposition, ...
0038     'Color', Gproperties.Gcol, ...
0039     'ToolBar', 'none', ...
0040     'MenuBar', 'none', ...
0041     'Resize', 'off', ...
0042     'Tag', 'LTPDAREpomainfig');
0043
0044 % Set mainfig callbacks
0045 set(mainfig, 'CloseRequestFcn', {'cb_mainfigClose', mainfig});
0046
0047 % Set Application data
0048 setappdata(mainfig, 'Gproperties', Gproperties);
0049 setappdata(mainfig, 'connection', []);
0050 setappdata(mainfig, 'Nconditions', 0);
0051 setappdata(mainfig, 'tableSelected', 1);
0052
0053 %% Add components
0054
0055 Tleft = Gproperties.Gborder/Gproperties.Gwidth;
0056 Theight = 1-2*Gproperties.Gborder/Gproperties.Gheight - 0.07;
0057 Tbottom = (1 - Gproperties.Gborder/Gproperties.Gheight - Theight);
0058 Twidth = 1-2*Gproperties.Gborder/Gproperties.Gwidth;
0059
0060 htab = uitabpanel(...
0061     'Parent', mainfig, ...
0062     'Style', 'lefttop', ...
0063     'Units', 'normalized', ...
0064     'Position', [Tleft Tbottom Twidth Theight], ...
0065     'FontSize', fontsize, ...
0066     'FrameBackgroundColor', Gproperties.Gcol, ...
0067     'FrameBorderType', 'etchedout', ...
0068     'Title', {'Connection', 'Submit', 'Query', 'Retrieve'}, ...%, 'Quick Look'}, ...
0069     'HorizontalAlignment', 'left', ...
0070     'FontWeight', 'bold', ...
0071     'TitleBackgroundColor', Gproperties.Gcol, ...
0072     'TitleForegroundColor', 'k', ...
0073     'TitleHighlightColor', [1 0 0], ...
0074     'PanelBackgroundColor', Gproperties.Gcol, ...
0075     'PanelBorderType', 'line');
0076
0077 hpanel = getappdata(htab, 'panels');
0078 setappdata(mainfig, 'panels', hpanel);
0079
0080 %----- Connection Panel -----
0081 cp = create_connection_panel(hpanel(1), Gproperties);
0082 %----- Submit Panel -----
0083 sp = create_submit_panel(hpanel(2), Gproperties);
0084 %----- Query Panel -----
0085 qp = create_query_panel(hpanel(3), Gproperties);
0086 %----- Query Panel -----
0087 rp = create_retrieve_panel(hpanel(4), Gproperties);
0088 %----- Quick Look Panel -----
0089 % rp = create_quicklook_panel(hpanel(5), Gproperties);
0090
0091 Ppos = get(mainfig, 'Position');
0092 %-- Status Txt
0093 Sleft = Gproperties.Gborder;
0094 Sbottom = Gproperties.Gborder + 10;
0095 Swidth = 60;
0096 Sheight = 20;
0097
0098 sth = uicontrol(mainfig, 'Style', 'text', ...
0099     'String', 'Status:', ...
0100     'FontSize', fontsize, ...
0101     'BackgroundColor', Gproperties.Gcol, ...
0102     'HorizontalAlignment', 'left', ...
0103     'Position', [Sleft Sbottom Swidth Sheight]);
0104 %-- Status bar
0105 pos = get(sth, 'Position');
0106 Sleft = 4*Gproperties.Gborder + pos(2);
0107 Swidth = Gproperties.Gwidth - Gproperties.Gborder - Sleft;
0108 Sheight = 30;
0109 Sbottom = Gproperties.Gborder;
0110 sth = uicontrol(mainfig, 'Style', 'text', ...
0111     'String', 'not connected', ...
0112     'FontSize', fontsize, ...

```

```

0113 'BackgroundColor', 'w',...
0114 'ForegroundColor', 'r',...
0115 'HorizontalAlignment', 'left',...
0116 'Max', 100,...
0117 'Position',[Sleft Sbottom Swidth Sheight],...
0118 'Tag', 'statusTxt');
0119
0120 %% Start the GUI
0121
0122 % Make the GUI visible.
0123 set(mainfig,'Visible','on')
0124
0125 h = findobj(mainfig, 'Tag', 'statusTxt');
0126 get(h, 'String')
0127
0128 end
0129
0130 %-----
0131 % Create the quick look panel
0132 %
0133 function panel = create_quicklook_panel(panel, Gproperties)
0134
0135 Ppos = get(panel, 'Position');
0136 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0137
0138 %-- Text string
0139 Sleft = 10;
0140 Sheight = 20;
0141 Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0142 Swidth = 200;
0143 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0144 sth = uicontrol(panel,'Style','text',...
0145 'String','LTPDA Objects',...
0146 'FontSize', fontsize,...
0147 'BackgroundColor', Gproperties.Gcol,...
0148 'HorizontalAlignment','center',...
0149 'Position',[Sleft Sbottom Swidth Sheight]);
0150
0151 %-- Workspace list of LTPDA objects
0152 Lleft = 10;
0153 Lheight = 350;
0154 Lbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Lheight;
0155 Lwidth = Swidth;
0156
0157 lbh = uicontrol(panel,'Style','listbox',...
0158 'String',{' '},...
0159 'Value',1,...
0160 'BackgroundColor', 'w',...
0161 'FontSize', fontsize,...
0162 'Max', 1000,...
0163 'Position',[Lleft Lbottom Lwidth Lheight],...
0164 'Tag', 'workspaceObjsListQL');
0165
0166 % Set callback
0167 set(lbh, 'Callback', {'cb_workspaceObjsListQL', lbh, mainfig});
0168
0169 % fill list
0170 objs = getWorkspaceObjs();
0171 setWorkspaceObjsListQL(objs);
0172
0173 %-- Refresh button
0174 pos = get(lbh, 'Position');
0175 Bleft = Lleft;
0176 Bheight = 20;
0177 Bwidth = 100;
0178 Bbottom = pos(2) - Bheight - 10;
0179 pbh = uicontrol(panel,'Style','pushbutton','String','Refresh list',...
0180 'Position',[Bleft Bbottom Bwidth Bheight ],...
0181 'Tag', 'refreshListBtnQL');
0182 set(pbh, 'Callback', {'cb_refreshObjsListQL', pbh, mainfig});
0183
0184
0185 %-- Info panel
0186 pos = get(lbh, 'Position');
0187 Sleft = 10 + pos(1) + pos(3);
0188 Sheight = 150;
0189 Sbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Sheight;
0190 Swidth = 550;
0191 iph = uicontrol(panel,'Style','edit',...
0192 'String',' ',...
0193 'FontSize', fontsize,...
0194 'BackgroundColor', 'w',...
0195 'ForegroundColor', 'b',...
0196 'HorizontalAlignment', 'left',...
0197 'Enable', 'on',...
0198 'Max', 1000,...

```



```

0199 'Position',[Sleft Sbottom Swidth Sheight],...
0200 'Tag', 'ltpdaObjInfoQL');
0201
0202 %-- Text string
0203 pos = get(iph, 'Position');
0204 Sheight = 20;
0205 Sbottom = pos(2) - Sheight - 20;
0206 Swidth = 300;
0207 sth = uicontrol(panel,'Style','text',...
0208 'String','Class Methods',...
0209 'FontSize', fontsize,...
0210 'BackgroundColor', Gproperties.Gcol,...
0211 'HorizontalAlignment','left',...
0212 'Position',[Sleft Sbottom Swidth Sheight]);
0213
0214 %-- List of class methods
0215 pos = get(lbh, 'Position');
0216 Lleft = pos(1) + pos(3) + 10;
0217 Lheight = 150;
0218 Lbottom = pos(2);
0219 Lwidth = 150;
0220
0221 lbh = uicontrol(panel,'Style','listbox',...
0222 'String',{'',''},...
0223 'Value',1,...
0224 'BackgroundColor','w',...
0225 'FontSize', fontsize,...
0226 'Max', 1000,...
0227 'Position',[Lleft Lbottom Lwidth Lheight],...
0228 'Tag', 'classMethodList');
0229
0230 % Set callback
0231 set(lbh, 'Callback', {'cb_classMethodList', lbh, mainfig});
0232
0233 % fill list
0234 fillClassMethodsList();
0235
0236 %----- Help panel
0237 pos = get(lbh, 'Position');
0238 Sleft = pos(1) + pos(3) + 10;
0239 Sheight = 150;
0240 Sbottom = pos(2);
0241 Swidth = 550;
0242 sth = uicontrol(panel,'Style','edit',...
0243 'String','',...
0244 'FontSize', fontsize,...
0245 'BackgroundColor','w',...
0246 'ForegroundColor','b',...
0247 'HorizontalAlignment','left',...
0248 'Enable','on',...
0249 'Max', 1000,...
0250 'Position',[Sleft Sbottom Swidth Sheight],...
0251 'Tag', 'methodInfoPanel');
0252
0253
0254 end
0255
0256 %-----
0257 % Create the retrieve panel
0258 %
0259 function panel = create_retrieve_panel(panel, Gproperties)
0260
0261 Ppos = get(panel, 'Position');
0262 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0263 ALLHEIGHT = 25;
0264
0265 %-- Text string
0266 Sleft = 10;
0267 Sheight = ALLHEIGHT;
0268 Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0269 Swidth = 300;
0270 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0271 Osth = uicontrol(panel,'Style','text',...
0272 'String','Retrieve object IDs',...
0273 'FontSize', fontsize,...
0274 'BackgroundColor', Gproperties.Gcol,...
0275 'HorizontalAlignment','left',...
0276 'Position',[Sleft Sbottom Swidth Sheight]);
0277
0278 %-- ID entry
0279 pos = get(Osth, 'Position');
0280 Sleft = 10;
0281 Sheight = 250;
0282 Swidth = 250;
0283 Sbottom = pos(2) - 20 - Sheight;
0284

```

```

0285 sth = uicontrol(panel,'Style','edit',...
0286     'String','',...
0287     'FontSize', fontsize,...
0288     'BackgroundColor', 'w',...
0289     'Max', 10,...
0290     'HorizontalAlignment', 'left',...
0291     'Position',[Sleft Sbottom Swidth Sheight],...
0292     'Tag', 'retrieveIDsTxt');
0293
0294 %-- Prefix entry
0295 pos = get(sth, 'Position');
0296 Sleft = 10;
0297 Sheight = ALLHEIGHT;
0298 Sbottom = pos(2) - 20 - Sheight;
0299 Swidth = 50;
0300 sth = uicontrol(panel,'Style','text',...
0301     'String','Prefix',...
0302     'FontSize', fontsize,...
0303     'BackgroundColor', Gproperties.Gcol,...
0304     'HorizontalAlignment', 'left',...
0305     'Position',[Sleft Sbottom Swidth Sheight]);
0306
0307 Sleft = Sleft + Swidth + 20;
0308 Sheight = ALLHEIGHT;
0309 Swidth = 150;
0310 sth = uicontrol(panel,'Style','edit',...
0311     'String','obj',...
0312     'FontSize', fontsize,...
0313     'BackgroundColor', 'w',...
0314     'HorizontalAlignment', 'left',...
0315     'Position',[Sleft Sbottom Swidth Sheight],...
0316     'Tag', 'objPrefixTxt');
0317
0318 %-- Append obj type
0319 pos = get(sth, 'Position');
0320 Sleft = 10;
0321 Swidth = 150;
0322 Sheight = ALLHEIGHT;
0323 Sbottom = pos(2) - 20 - Sheight;
0324 cbh = uicontrol(panel,'Style','checkbox',...
0325     'String','Append object type',...
0326     'FontSize', fontsize,...
0327     'BackgroundColor', Gproperties.Gcol,...
0328     'Value',1,'Position',[Sleft Sbottom Swidth Sheight],...
0329     'Tag', 'appendObjTypeChk');
0330
0331 %-- Import Btn
0332
0333 Bleft = 10;
0334 Bheight = ALLHEIGHT;
0335 Bwidth = 100;
0336 Bbottom = Ppos(2) + 10;
0337 pbh = uicontrol(panel,'Style','pushbutton','String','Import',...
0338     'Position',[Bleft Bbottom Bwidth Bheight],...
0339     'FontSize', fontsize,...
0340     'Fontweight','bold',...
0341     'Tag', 'importBtn');
0342 set(pbh, 'Callback',{ 'cb_importBtn', pbh, mainfig});
0343 set(pbh, 'TooltipString','Retrieve objects to MATLAB workspace');
0344
0345 %-- Save Btn
0346
0347 Bleft = 120;
0348 Bheight = ALLHEIGHT;
0349 Bwidth = 100;
0350 Bbottom = Ppos(2) + 10;
0351 pbh = uicontrol(panel,'Style','pushbutton','String','Save',...
0352     'Position',[Bleft Bbottom Bwidth Bheight],...
0353     'FontSize', fontsize,...
0354     'Fontweight','bold',...
0355     'Tag', 'saveBtn');
0356 set(pbh, 'Callback',{ 'cb_saveBtn', pbh, mainfig});
0357 set(pbh, 'TooltipString','Retrieve objects to disk');
0358
0359
0360
0361 end
0362
0363 %-----
0364 % Create the Query panel
0365 %
0366 function panel = create_query_panel(panel, Gproperties)
0367
0368 Ppos = get(panel, 'Position');
0369 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0370

```

```

0371 %-- Build conditions
0372 buildConditions(panel);
0373
0374 end
0375
0376 %-----
0377 % Create the submit panel
0378 %
0379 function panel = create_submit_panel(panel, Gproperties)
0380
0381 Ppos      = get(panel, 'Position');
0382 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0383 ALLHEIGHT = 25;
0384 %-----
0385 %-- Text string
0386 Sleft     = 10;
0387 Sheight   = ALLHEIGHT;
0388 Sbottom   = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0389 Swidth     = 150;
0390 fontsize  = getappdata(0, 'ltpda_repo_gui_fontsize');
0391 sth = uicontrol(panel, 'Style', 'text', ...
0392     'String', 'LTPDA Objects', ...
0393     'FontSize', fontsize, ...
0394     'BackgroundColor', Gproperties.Gcol, ...
0395     'HorizontalAlignment', 'center', ...
0396     'Position', [Sleft Sbottom Swidth Sheight]);
0397
0398 %-----
0399 %-- Workspace list of LTPDA objects
0400 Lleft     = 10;
0401 Lheight   = 350;
0402 Lbottom   = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Lheight;
0403 Lwidth     = Swidth;
0404
0405 lbh = uicontrol(panel, 'Style', 'listbox', ...
0406     'String', {''}, ...
0407     'Value', 1, ...
0408     'BackgroundColor', 'w', ...
0409     'FontSize', fontsize, ...
0410     'Max', 1000, ...
0411     'Position', [Lleft Lbottom Lwidth Lheight], ...
0412     'Tag', 'workspaceObjsList');
0413
0414 % Set callback
0415 set(lbh, 'Callback', {'cb_workspaceObjsList', lbh, mainfig});
0416
0417 % fill list
0418 objs = getWorkspaceObjs();
0419 setWorkspaceObjsList(objs);
0420
0421 %-----
0422 %-- Refresh button
0423 pos      = get(lbh, 'Position');
0424 Bleft    = Lleft;
0425 Bheight  = ALLHEIGHT;
0426 Bwidth   = 100;
0427 Bbottom  = pos(2) - Bheight - 10;
0428 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Refresh list', ...
0429     'Position', [Bleft Bbottom Bwidth Bheight], ...
0430     'Tag', 'refreshListBtn');
0431 set(pbh, 'Callback', {'cb_refreshObjsList', pbh, mainfig});
0432
0433 %-----
0434 %-- Info panel
0435 % pos = get(lbh, 'Position');
0436 % Sleft = 10 + pos(1) + pos(3);
0437 % Sheight = 150;
0438 % Sbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Sheight;
0439 % Swidth = 500;
0440 % sth = uicontrol(panel, 'Style', 'text', ...
0441 %     'String', '', ...
0442 %     'FontSize', fontsize, ...
0443 %     'BackgroundColor', 'w', ...
0444 %     'ForegroundColor', 'b', ...
0445 %     'HorizontalAlignment', 'left', ...
0446 %     'Position', [Sleft Sbottom Swidth Sheight], ...
0447 %     'Tag', 'ltpdaObjInfo');
0448
0449 %-----
0450 %-- experiment title
0451 pos      = get(sth, 'Position');
0452 Sleft    = 20 + pos(1) + pos(3);
0453 Sheight  = ALLHEIGHT;
0454 Swidth   = 150;
0455 % Sbottom = pos(2) - 20 - Sheight;

```

```

0457 Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
0458
0459 sth = uicontrol(panel, 'Style', 'text', ...
0460     'String', 'Experiment Title', ...
0461     'FontSize', fontsize, ...
0462     'BackgroundColor', Gproperties.Gcol, ...
0463     'HorizontalAlignment', 'left', ...
0464     'FontWeight', 'bold', ...
0465     'ForegroundColor', 'r', ...
0466     'Position', [Sleft Sbottom Swidth Sheight]);
0467 Tleft = Sleft + Swidth + 20;
0468 Theight = ALLHEIGHT;
0469 Twidth = 380;
0470 Tbottom = Sbottom;
0471
0472 sth = uicontrol(panel, 'Style', 'edit', ...
0473     'String', '', ...
0474     'FontSize', fontsize, ...
0475     'BackgroundColor', 'w', ...
0476     'HorizontalAlignment', 'left', ...
0477     'Position', [Tleft Tbottom Twidth Theight], ...
0478     'Tag', 'experimentTitle');
0479
0480 ttstr = 'A short title for the experiment from which the data originates.';
0481 set(sth, 'Tooltipstring', ttstr)
0482
0483 %-----
0484 %-- experiment description
0485 Sheight = ALLHEIGHT;
0486 Tbottom = Tbottom - 20 - ALLHEIGHT;
0487
0488 sth = uicontrol(panel, 'Style', 'text', ...
0489     'String', 'Experiment description', ...
0490     'FontSize', fontsize, ...
0491     'FontWeight', 'bold', ...
0492     'ForegroundColor', 'r', ...
0493     'BackgroundColor', Gproperties.Gcol, ...
0494     'HorizontalAlignment', 'left', ...
0495     'Position', [Sleft Tbottom Swidth Sheight]);
0496
0497 Extra = 60;
0498 Tbottom = Tbottom + 20 - Theight - Extra;
0499 Theight = ALLHEIGHT + Extra;
0500 sth = uicontrol(panel, 'Style', 'edit', ...
0501     'String', '', ...
0502     'FontSize', fontsize, ...
0503     'BackgroundColor', 'w', ...
0504     'Max', 100, ...
0505     'HorizontalAlignment', 'left', ...
0506     'Position', [Tleft Tbottom Twidth Theight], ...
0507     'Tag', 'experimentDescription');
0508
0509 ttstr = 'A description of the experiment from which the data originates.';
0510 set(sth, 'Tooltipstring', ttstr)
0511
0512 %-----
0513 %-- analysis description
0514 Tbottom = Tbottom - 20 - Sheight;
0515
0516 sth = uicontrol(panel, 'Style', 'text', ...
0517     'String', 'Analysis description', ...
0518     'FontSize', fontsize, ...
0519     'FontWeight', 'bold', ...
0520     'ForegroundColor', 'r', ...
0521     'BackgroundColor', Gproperties.Gcol, ...
0522     'HorizontalAlignment', 'left', ...
0523     'Position', [Sleft Tbottom Swidth Sheight]);
0524
0525 Extra = ALLHEIGHT;
0526 Tbottom = Tbottom - Extra;
0527 Theight = ALLHEIGHT + Extra;
0528 sth = uicontrol(panel, 'Style', 'edit', ...
0529     'String', '', ...
0530     'FontSize', fontsize, ...
0531     'BackgroundColor', 'w', ...
0532     'Max', 100, ...
0533     'HorizontalAlignment', 'left', ...
0534     'Position', [Tleft Tbottom Twidth Theight], ...
0535     'Tag', 'analysisDescription');
0536
0537 ttstr = 'A description of the analysis performed on the data.';
0538 set(sth, 'Tooltipstring', ttstr)
0539
0540 %-----
0541 %-- quantity
0542 Tbottom = Tbottom - 20 - Sheight;

```

```

0543
0544 sth = uicontrol(panel,'Style','text',...
0545     'String','Quantity',...
0546     'FontSize', fontsize,...
0547     'BackgroundColor', Gproperties.Gcol,...
0548     'HorizontalAlignment','left',...
0549     'Position',[Sleft Tbottom Swidth Sheight]);
0550
0551 Theight = ALLHEIGHT;
0552 Tbottom = Tbottom;
0553 sth = uicontrol(panel,'Style','edit',...
0554     'String','',...
0555     'FontSize', fontsize,...
0556     'BackgroundColor','w',...
0557     'Max', 1,...
0558     'HorizontalAlignment','left',...
0559     'Position',[Tleft Tbottom Twidth Theight],...
0560     'Tag','quantity');
0561
0562 ttstr = 'The physical quantity that the data represents.';
0563 set(sth, 'Tooltipstring', ttstr)
0564
0565 %-----
0566 %-- keywords
0567 Tbottom = Tbottom - 20 - Theight;
0568
0569 sth = uicontrol(panel,'Style','text',...
0570     'String','Keywords',...
0571     'FontSize', fontsize,...
0572     'BackgroundColor', Gproperties.Gcol,...
0573     'HorizontalAlignment','left',...
0574     'Position',[Sleft Tbottom Swidth Sheight]);
0575
0576 Extra = 20;
0577 Theight = ALLHEIGHT + Extra;
0578 Tbottom = Tbottom - Extra;
0579 sth = uicontrol(panel,'Style','edit',...
0580     'String','',...
0581     'FontSize', fontsize,...
0582     'BackgroundColor','w',...
0583     'Max', 1,...
0584     'HorizontalAlignment','left',...
0585     'Position',[Tleft Tbottom Twidth Theight],...
0586     'Tag','keywords');
0587
0588 ttstr = 'A comma-delimited list of keywords.';
0589 set(sth, 'Tooltipstring', ttstr)
0590
0591 %-----
0592 %-- reference ids
0593 Tbottom = Tbottom - Theight;
0594
0595 sth = uicontrol(panel,'Style','text',...
0596     'String','Reference IDs',...
0597     'FontSize', fontsize,...
0598     'BackgroundColor', Gproperties.Gcol,...
0599     'HorizontalAlignment','left',...
0600     'Position',[Sleft Tbottom Swidth Sheight]);
0601
0602 Theight = ALLHEIGHT;
0603 Tbottom = Tbottom;
0604 sth = uicontrol(panel,'Style','edit',...
0605     'String','',...
0606     'FontSize', fontsize,...
0607     'BackgroundColor','w',...
0608     'Max', 1,...
0609     'HorizontalAlignment','left',...
0610     'Position',[Tleft Tbottom Twidth Theight],...
0611     'Tag','referenceIDs');
0612
0613 ttstr = 'List any object IDs that are relevant to this/these results.';
0614 set(sth, 'Tooltipstring', ttstr)
0615
0616 %-----
0617 %-- additional comments
0618 Tbottom = Tbottom - 20 - Theight;
0619
0620 sth = uicontrol(panel,'Style','text',...
0621     'String','Additional Comments',...
0622     'FontSize', fontsize,...
0623     'BackgroundColor', Gproperties.Gcol,...
0624     'HorizontalAlignment','left',...
0625     'Position',[Sleft Tbottom Swidth Sheight]);
0626
0627 Extra = 30;
0628 Tbottom = Tbottom - Extra;

```



```

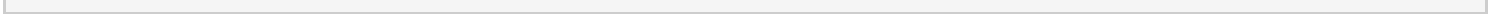
0629 Theight = ALLHEIGHT + Extra;
0630 sth = uicontrol(panel, 'Style', 'edit', ...
0631     'String', '', ...
0632     'FontSize', fontsize, ...
0633     'BackgroundColor', 'w', ...
0634     'Max', 100, ...
0635     'HorizontalAlignment', 'left', ...
0636     'Position', [Tleft Tbottom Twidth Theight], ...
0637     'Tag', 'additionalComments');
0638
0639 ttstr = 'Anything else you want to say about the objects being submitted.';
0640 set(sth, 'Tooltipstring', ttstr)
0641
0642 %-----
0643 %-- additional authors
0644 Tbottom = Tbottom - Theight;
0645
0646 sth = uicontrol(panel, 'Style', 'text', ...
0647     'String', 'Additional Authors', ...
0648     'FontSize', fontsize, ...
0649     'BackgroundColor', Gproperties.Gcol, ...
0650     'HorizontalAlignment', 'left', ...
0651     'Position', [Sleft Tbottom Swidth Sheight]);
0652
0653 Extra = 0;
0654 Tbottom = Tbottom - Extra;
0655 Theight = ALLHEIGHT + Extra;
0656 sth = uicontrol(panel, 'Style', 'edit', ...
0657     'String', '', ...
0658     'FontSize', fontsize, ...
0659     'BackgroundColor', 'w', ...
0660     'Max', 100, ...
0661     'HorizontalAlignment', 'left', ...
0662     'Position', [Tleft Tbottom Twidth Theight], ...
0663     'Tag', 'additionalAuthors');
0664
0665 ttstr = 'Who else helped creating these objects?';
0666 set(sth, 'Tooltipstring', ttstr)
0667
0668 %-----
0669 %-- Submit Btn
0670
0671 pos      = get(lbh, 'Position');
0672 Bleft    = Lleft;
0673 Bheight  = ALLHEIGHT;
0674 Bwidth   = 100;
0675 Bbottom  = Ppos(2) + 10;
0676 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Submit', ...
0677     'Position', [Bleft Bbottom Bwidth Bheight], ...
0678     'FontSize', fontsize, ...
0679     'Fontweight', 'bold', ...
0680     'Tag', 'submitBtn');
0681 set(pbh, 'Callback', {'cb_submitBtn', pbh, mainfig});
0682
0683
0684
0685 end
0686
0687
0688 %-----
0689 % Create the server connection panel
0690 %
0691 function panel = create_connection_panel(panel, Gproperties)
0692
0693
0694 % %-- Panel
0695 % Pleft = Gproperties.Gborder/Gproperties.Gheight;
0696 % Pbottom = Gproperties.Gborder/Gproperties.Gheight;
0697 % Pwidth = 1-2*Gproperties.Gborder/Gproperties.Gwidth;
0698 % Pheight = 0.4;
0699 %
0700 % hp = uipanel('Parent', mainfig, ...
0701 %     'Title', 'Server Connection', ...
0702 %     'FontSize', 14, ...
0703 %     'BackgroundColor', Gproperties.Gcol, ...
0704 %     'Position', [Pleft Pbottom Pwidth Pheight], ...
0705 %     'BorderType', 'etchedout', ...
0706 %     'HighlightColor', [0.1 0.9 0.1]);
0707
0708 Ppos      = get(panel, 'Position');
0709 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0710
0711 %-- Server selection
0712 Sleft     = 10;
0713 Sbottom   = Ppos(2) + Ppos(4) - 40-Gproperties.Gborder;
0714 Swidth    = 150;

```

```

0715 Sheight = 25;
0716 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0717 servers = getappdata(0, 'ltpda_repo_servers');
0718
0719 pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0720     'String', servers, ...
0721     'FontSize', fontsize, ...
0722     'BackgroundColor', Gproperties.Gcol, ...
0723     'Value', 1, 'Position', [Sleft Sbottom Swidth Sheight], ...
0724     'Tag', 'serverSelect');
0725 set(pmh, 'Callback', {'cb_serverSelect', pmh, mainfig});
0726
0727 %-- Server Txt entry
0728 Tleft = Sleft + Swidth + 10;
0729 Tbottom = Sbottom;
0730 Twidth = 200;
0731 Theight = Sheight;
0732
0733 eth = uicontrol(panel, 'Style', 'edit', ...
0734     'String', 'localhost', ...
0735     'FontSize', fontsize, ...
0736     'BackgroundColor', 'w', ...
0737     'Position', [Tleft Tbottom Twidth Theight], ...
0738     'Tag', 'serverTxt');
0739
0740 %-- Text string
0741 sth = uicontrol(panel, 'Style', 'text', ...
0742     'String', 'Server hostname', ...
0743     'FontSize', fontsize, ...
0744     'BackgroundColor', Gproperties.Gcol, ...
0745     'HorizontalAlignment', 'left', ...
0746     'Position', [Tleft+Twidth+10 Tbottom 120 Theight]);
0747
0748 %-- Database selection
0749 pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0750     'String', {'ltpda_test', 'utp', 'test'}, ...
0751     'FontSize', fontsize, ...
0752     'BackgroundColor', Gproperties.Gcol, ...
0753     'Value', 1, 'Position', [Sleft Sbottom-Sheight-10 Swidth Sheight], ...
0754     'Tag', 'dbSelect');
0755 set(pmh, 'Callback', {'cb_dbSelect', pmh, mainfig});
0756
0757 %-- Database Txt entry
0758 eth = uicontrol(panel, 'Style', 'edit', ...
0759     'String', 'ltpda_test', ...
0760     'FontSize', fontsize, ...
0761     'BackgroundColor', 'w', ...
0762     'Position', [Tleft Tbottom-Theight-10 Twidth Theight], ...
0763     'Tag', 'databaseTxt');
0764
0765 %-- Text string
0766 sth = uicontrol(panel, 'Style', 'text', ...
0767     'String', 'Database', ...
0768     'FontSize', fontsize, ...
0769     'BackgroundColor', Gproperties.Gcol, ...
0770     'HorizontalAlignment', 'left', ...
0771     'Position', [Tleft+Twidth+10 Tbottom-Theight-10 120 Theight]);
0772
0773 %-- Available DBs button
0774 pos = get(sth, 'Position');
0775 Bleft = pos(1) + pos(3) + 20;
0776 Bbottom = pos(2);
0777 Bwidth = 100;
0778 Bheight = Sheight;
0779 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Get DBs', ...
0780     'Position', [Bleft Bbottom Bwidth Bheight], ...
0781     'Tag', 'getDbBtn');
0782 set(pbh, 'Callback', {'cb_getDBs', pbh, mainfig});
0783
0784 %-- Login Btn
0785 pos = get(pmh, 'Position');
0786 Bleft = pos(1);
0787 Bbottom = pos(2) - pos(4) - 20;
0788 Bwidth = 100;
0789 Bheight = Sheight;
0790
0791 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Connect', ...
0792     'Position', [Bleft Bbottom Bwidth Bheight], ...
0793     'Tag', 'connectBtn');
0794 set(pbh, 'Callback', {'cb_connectBtn', pbh, mainfig});
0795
0796
0797
0798
0799 end
0800

```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [sqlResultsGUI.m](#)

sqlResultsGUI

PURPOSE [^]

SQLRESULTSGUI shows the results of a query in a new window.

SYNOPSIS [^]

```
function sqlResultsGUI(results, fields, query)
```

DESCRIPTION [^]

```
SQLRESULTSGUI shows the results of a query in a new window.
sqlResultsGUI(results, fields)
M Hewitson 09-07-07
$Id: sqlResultsGUI.m,v 1.10 2008/03/11 15:51:48 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [createTable](#) createTable - create nice-looking table based on javax.swing.JTable

This function is called by:

SUBFUNCTIONS [^]

 [function qtxtCB\(hObject, eventdata, query\)](#)
 [function exportBtn_Callback\(hObject, eventdata, gui, results, fields\)](#)
 [function resizeMain\(hObject, eventdata, leftMargin, bottomMargin, topMargin\)](#)

SOURCE CODE [^]

```
0001 function sqlResultsGUI(results, fields, query)
0002
0003 % SQLRESULTSGUI shows the results of a query in a new window.
0004 %
0005 % sqlResultsGUI(results, fields)
0006 %
0007 % M Hewitson 09-07-07
0008 %
0009 % $Id: sqlResultsGUI.m,v 1.10 2008/03/11 15:51:48 hewitson Exp $
0010 %
0011
0012 leftMargin    = 10;
0013 bottomMargin  = 100;
0014 topMargin     = 20;
0015
0016 btnHeight     = 20;
0017 btnWidth      = 60;
0018 btnGap        = 20;
0019
0020 Color         = [240 240 240]/255;
0021 Height        = 600;
0022 Width         = 800;
0023
0024 Screen        = get(0, 'screensize');
0025 Position      = [Screen(3)/2 - Width/2 Screen(4)/2-Height/2 Width Height];
0026
0027 % create figure
```

```

0028 gui.main = figure('HandleVisibility','on',...
0029     'IntegerHandle','off',...
0030     'Menubar','none',...
0031     'NumberTitle','off',...
0032     'Name','Query Results',...
0033     'Tag','main',...
0034     'Color',Color,...
0035     'Resize','on',...
0036     'Position',Position,...
0037     'ResizeFcn',{@resizeMain, leftMargin, bottomMargin, topMargin});
0038
0039 % add query string
0040 Tleft = leftMargin;
0041 Twidth = Width - 2*leftMargin;
0042 Theight = 80;
0043 Tbottom = Position(4) - Theight - 20;
0044 sth = uicontrol(gui.main,'Style','edit',...
0045     'String',query,...
0046     'FontSize',14,...
0047     'BackgroundColor','w',...
0048     'Max',100,...
0049     'HorizontalAlignment','left',...
0050     'Tag','QueryString',...
0051     'Enable','on',...
0052     'Position',[Tleft Tbottom Twidth Theight],...
0053     'Callback',{@txtCB, query});
0054
0055 setappdata(gui.main, 'qstring', sth);
0056
0057 if size(fields,2) ~= size(results,2)
0058     fields = cell(1,size(results,2));
0059 end
0060
0061 % In MATLAB 2008a we can use a proper uitable:
0062 v = ver('MATLAB');
0063 if strcmp(v.Version, '7.6')
0064     t = uitable(gui.main, 'Data', results, ...
0065         'ColumnName', fields,...
0066         'units','normalized',...
0067         'Position', [.01,.08,1-0.02,.73]);
0068 else
0069     % Otherwise, we use the createTable function which uses a JTable
0070     [t,b] = createTable(gui.main, fields, results, false, ...
0071         'AutoResizeMode',javax.swing.JTable.AUTO_RESIZE_OFF,...
0072         'Editable',false, 'Position', [.01,.08,1-0.02,.73]);
0073 end
0074
0075 setappdata(gui.main, 'table', t);
0076
0077 % add export button
0078 btn = uicontrol(gui.main,...
0079     'Style','pushbutton',...
0080     'String','Export',...
0081     'Tag','exportBtn',...
0082     'Callback',{@exportBtn_Callback, gui, results, fields},...
0083     'Position',[leftMargin leftMargin btnWidth btnHeight], 'Visible', 'off');
0084
0085 get(btn)
0086
0087 %-----
0088 % Callbacks
0089
0090 % callback to reset the query string in case the user deletes it
0091 function txtCB(hObject, eventdata, query)
0092
0093 set(hObject, 'String', query);
0094
0095 %-----
0096 % export button
0097 function exportBtn_Callback(hObject, eventdata, gui, results, fields)
0098
0099 % get table
0100 t = getappdata(gui.main, 'table');
0101 table = get(t.Table);
0102
0103 rows = table.SelectedRows + 1;
0104 cols = table.SelectedColumns + 1;
0105 res = results(rows, cols);
0106 nResults = size(res, 1);
0107
0108 % get file to export to
0109 [filename, pathname, filteridx] = uiputfile('*.txt', 'Select export file');
0110
0111 fd = fopen(filename, 'w+');

```

```

0114
0115 % header
0116 fprintf(fd, '# ');
0117 fprintf(fd, '%s\t\t', fields{cols});
0118 fprintf(fd, '\n');
0119
0120
0121 for j=1:nResults
0122     for k=1:size(res,2)
0123         r = res{j,k};
0124         if ischar(r)
0125             fprintf(fd, '%s\t\t', char(r));
0126         elseif isnumeric(r)
0127             fprintf(fd, '%f\t\t', r);
0128         else
0129             error('### Unknown data format.');
```

```

0130         end
0131     end
0132     fprintf(fd, '\n');
0133 end
0134 fclose(fd);
0135
0136 edit(filename);
0137
0138 %-----
0139 % Resize Main figure
0140 function resizeMain(hObject, eventdata, leftMargin, bottomMargin, topMargin)
0141
0142 % resize table
0143 s = getappdata(hObject, 'qstring');
0144 ps = get(s, 'Position');
0145 t = getappdata(hObject, 'table');
0146 fs = get(hObject, 'Position');
0147 pos = get(t, 'Position');
0148
0149 % set(t, 'Position', [pos(1) pos(2) fs(3)-2*leftMargin fs(4)-bottomMargin-ps(4)]);
0150 set(s, 'Position', [ps(1) fs(4)-ps(4)-20 fs(3)-2*leftMargin ps(4)]);
0151
0152
0153 % END

```

uitabpanel

PURPOSE ^

UITABPANEL adds a group of tabbed uipanel container objects to a figure.

SYNOPSIS ^

function hh = uitabpanel(varargin)

DESCRIPTION ^

UITABPANEL adds a group of tabbed uipanel container objects to a figure. The user can click on a tab with the left mouse button to select it. One tab is always selected, and appears on top of all the orthers. The syntax is the same as UIPANEL except the following property changes:

Property	Values
Title	a cell array of strings specifying the title of each tab
String	(same as Title)
TitlePosition	the location of tabs in relation of the visible panel, the value is the same as UIPANEL
Style	(same as TitlePosition)
FrameBackgroundColor	the background color for the tabpanel frame
FrameBorderType	the border type for the tabpanel frame
PanelBackgroundColor	the background color for the active tab/panel
TitleHighlightColor	the highlight color for the active tab
TitleForegroundColor	the foreground color for tab titiles
TitleBackgroundColor	the background color for the inactive tabs
SelectedItem	index specifying the currently active tab

A yet another type of UITABPANEL is a group of popup panels, which toggles open and close of a panel when its titile is clicked. Additional properties for popup tabpanel are:

Property	Values
Style	'popup'
PanelHeights	a numeric array specfying the height in characters of each popup panel
PanelBorderType	the border type for the popup panels
TitleHighlightColor	(not applicable)

By defining 'CreateFcn', MATLAB will build the group of panels at creation of UITABPANEL. An array of handles of the UIPANELs can be retrieved by calling getappdata(hTabpanel,'panels').

Example:
htab = uitabpanel('title',{'1','2','3'})
hpanel = getappdata(htab,'panels')

See also uitabdemo, uipanel, uicontrol, set, get.

Adapted for LTPDA by M Hewitson

\$Id: uitabpanel.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [repopgui](#) REPOGUI a GUI to interact with an LTPDA Repository.

SUBFUNCTIONS ^

 `function PopupTabCbK(hobj,evdt,indx)`
 `function PopupTabSliderCbK(hobj,evdt)`
 `function PopupTabEnableAllCbK(hobj,evdt,cstr)`
 `function PopupTabResizeCbK(hobj,evdt)`
 `function TopBottomTabCbK(hobj,evdt,indx)`
 `function TopBottomTabResizeCbK(hobj,evdt)`

SOURCE CODE [^]

```

0001
0002 function hh = uitabpanel(varargin)
0003
0004 % UITABPANEL adds a group of tabbed uipanel container objects to a figure.
0005 % The user can click on a tab with the left mouse button to select it.
0006 % One tab is always selected, and appears on top of all the others.
0007 % The syntax is the same as UIPANEL except the following property changes:
0008 %
0009 % Property          Values
0010 % -----
0011 % Title             a cell array of strings specifying the title of
0012 %                   each tab
0013 % String            (same as Title)
0014 % TitlePosition     the location of tabs in relation of the visible
0015 %                   panel, the value is the same as UIPANEL
0016 % Style             (same as TitlePosition)
0017 % FrameBackgroundColor the background color for the tabpanel frame
0018 % FrameBorderType   the border type for the tabpanel frame
0019 % PanelBackgroundColor the background color for the active tab/panel
0020 % TitleHighlightColor the highlight color for the active tab
0021 % TitleForegroundColor the foreground color for tab titles
0022 % TitleBackgroundColor the background color for the inactive tabs
0023 % SelectedItem      index specifying the currently active tab
0024 %
0025 % A yet another type of UITABPANEL is a group of popup panels, which
0026 % toggles open and close of a panel when its title is clicked. Additional
0027 % properties for popup tabpanel are:
0028 %
0029 % Property          Values
0030 % -----
0031 % Style             'popup'
0032 % PanelHeights      a numeric array specifying the height in
0033 %                   characters of each popup panel
0034 % PanelBorderType   the border type for the popup panels
0035 % TitleHighlightColor (not applicable)
0036 %
0037 % By defining 'CreateFcn', MATLAB will build the group of panels at
0038 % creation of UITABPANEL. An array of handles of the UIPANELS can
0039 % be retrieved by calling getappdata(hTabpanel,'panels').
0040 %
0041 % Example:
0042 %   htab = uitabpanel('title',{'1','2','3'})
0043 %   hpanel = getappdata(htab,'panels')
0044 %
0045 % See also uitabdemo, uipanel, uicontrol, set, get.
0046 %
0047 % Adapted for LTPDA by M Hewitson
0048 %
0049 % $Id: uitabpanel.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0050 %
0051 %
0052
0053
0054 % Author: Shiyong Zhao (zhao@arch.ums1.edu)
0055 % Version: 1.0
0056 % First created: May 02, 2006
0057 % Last modified: May 20, 2006
0058
0059 tag = '';
0060 units = 'normalized';
0061 position = [0,0,1,1];
0062 fontweight = 'normal';
0063 [style,type] = deal('lefttop',2);
0064 visible = 'on';
0065
0066 foregroundcolor = [1,1,1];
0067 backgroundcolor = [0.7725,0.8431,0.9608];
0068 panelbackgroundcolor = [0.4314,0.5882,0.8431];
0069 titlehighlightcolor = [1,0.8,0];
0070

```

```

0071 horizontalalignment = 'center';
0072 [createfcn,deletefcn,resizefcn] = deal({}, {}, {});
0073
0074 active = 0;
0075 styles = {'popup', 'lefttop', 'righttop', 'leftbottom', 'rightbottom', ...
0076 'centertop', 'centerbottom'};
0077
0078 for k=1:2:nargin
0079     property = lower(varargin{k});
0080     value = varargin{k+1};
0081     switch property
0082     case {'style', 'tabposition', 'titleposition'}
0083         style = value;
0084         type = strmatch(lower(style), styles);
0085         if isempty(type)
0086             error(['Bad value for Style/TitlePosition property: ' style '.']);
0087         end
0088     case 'fontsize'
0089         fontsize = value;
0090     case {'title', 'string'}
0091         string = value;
0092         ntab = length(string);
0093     case {'frameposition', 'position'}
0094         position = value;
0095     case {'framebackgroundcolor', 'backgroundcolor'}
0096         backgroundcolor = value;
0097     case {'framebordertype', 'bordertype'}
0098         bordertype = value;
0099     case {'panelheights', 'heights'}
0100         panelheight = value;
0101     case {'titleforegroundcolor', 'foregroundcolor'}
0102         foregroundcolor = value;
0103     case {'selecteditem', 'active'}
0104         active = value;
0105     otherwise
0106         eval([property '= value;']);
0107     end
0108 end
0109
0110 if ~exist('parent', 'var')
0111     parent = gcf;
0112 end
0113 if ~exist('margins', 'var')
0114     if type==1
0115         margins = {[1.5, 0.75, 1.5, 1], 'characters'};
0116     else
0117         margins = {1, 'pixels'};
0118     end
0119 end
0120 if ~iscell(margins)
0121     margins = {margins, units};
0122 end
0123
0124 if ~exist('bordertype', 'var')
0125     if type==1
0126         bordertype = 'etchedin';
0127     else
0128         bordertype = 'none';
0129     end
0130 end
0131
0132 if ~exist('titlebackgroundcolor', 'var')
0133     titlebackgroundcolor = 0.95*panelbackgroundcolor;
0134 end
0135
0136 cmenu = uicontextmenu;
0137 uimenu(cmenu, 'Label', 'Goto Tab');
0138 uimenu(cmenu, 'Separator', 'on');
0139
0140 htab = uipanel( ...
0141     'Parent', parent, ...
0142     'Units', units, ...
0143     'Position', position, ...
0144     'DeleteFcn', deletefcn, ...
0145     'ResizeFcn', resizefcn, ...
0146     'BackgroundColor', backgroundcolor, ...
0147     'BorderType', bordertype, ...
0148     'UIContextMenu', cmenu, ...
0149     'Visible', visible, ...
0150     'FontSize', fontsize, ...
0151     'Tag', tag);
0152 set(htab, 'Units', 'pixels');
0153 tabpos = get(htab, 'Position');
0154
0155 status = uicontainer( ...
0156     'Parent', htab, ...

```

```

0157 'BackgroundColor', backgroundcolor, ...
0158 'Units', margins{2}, ...
0159 'Position', [1,1,1,1], ...
0160 'UIContextMenu', cmenu, ...
0161 'Visible', 'off');
0162 set(status, 'Units', 'pixels');
0163 charsz = get(status, 'Position');
0164 margins = [charsz(1:2), charsz(1:2)].*margins{1};
0165 set(status, 'Units', 'characters');
0166 set(status, 'Position', [1,1,1,1]);
0167 set(status, 'Units', 'pixels');
0168 charsz = get(status, 'Position');
0169 titleheight = 2*charsz(4);
0170
0171 if type==1
0172 %-----
0173 % PopupTab
0174 %-----
0175 panelpos = tabpos;
0176 titlepos = [margins(1), tabpos(2)+tabpos(4)-titleheight-margins(2), ...
0177     tabpos(3)-sum(margins([1,3])), titleheight];
0178 ttextpos = [0.05, 0.1, 0.9, 0.76];
0179
0180 if ~exist('panelheight', 'var')
0181     panelheight = ones(1, ntab);
0182 end
0183 if ~exist('panelbordertype', 'var')
0184     panelbordertype = 'line';
0185 end
0186
0187 for k=1:ntab
0188     title(k) = uicontrol( ...
0189         'Parent', htab, ...
0190         'Units', 'pixels', ...
0191         'Position', titlepos, ...
0192         'ForegroundColor', foregroundcolor, ...
0193         'BackgroundColor', titlebackgroundcolor, ...
0194         'FontWeight', fontweight, ...
0195         'FontSize', fontsize, ...
0196         'HorizontalAlignment', horizontalalignment, ...
0197         'Callback', {@PopupTabCbK, k}, ...
0198         'Style', 'toggle', ...
0199         'String', string{k}, ...
0200         'Value', 0, ...
0201         'Visible', 'on');
0202
0203     panel(k) = uipanel( ...
0204         'Parent', htab, ...
0205         'Units', 'pixels', ...
0206         'Position', titlepos+[0,0,0, panelheight(k)*charsz(4)], ...
0207         'BackgroundColor', panelbackgroundcolor, ...
0208         'BorderType', panelbordertype, ...
0209         'Visible', 'off');
0210
0211     uimenu(cmenu, ...
0212         'Label', string{k}, ...
0213         'Callback', {@PopupTabCbK, k});
0214 end
0215
0216 uimenu(cmenu, 'Separator', 'on');
0217 uimenu(cmenu, ...
0218     'Label', 'Expand All', ...
0219     'Callback', {@PopupTabEnableAllCbK, 'on'});
0220 uimenu(cmenu, ...
0221     'Label', 'Collapse All', ...
0222     'Callback', {@PopupTabEnableAllCbK, 'off'});
0223
0224 sliderpos = [tabpos(3)-14, 2, 15, tabpos(4)-3];
0225 slider = uicontrol( ...
0226     'Parent', htab, ...
0227     'Units', 'pixels', ...
0228     'Position', sliderpos, ...
0229     'BackgroundColor', panelbackgroundcolor, ...
0230     'Style', 'slider', ...
0231     'Callback', @PopupTabSliderCbK, ...
0232     'Value', 0, ...
0233     'Min', -1, ...
0234     'Max', 0, ...
0235     'Visible', 'off');
0236
0237 statuspos = titlepos;
0238 statuspos([2,4]) = [margins(4), panelpos(4)-(ntab+1)*(margins(4)+titlepos(4))];
0239 set(status, 'Position', statuspos);
0240 PopupTabCbK(title(1), [], active);
0241 set(status, 'ResizeFcn', @PopupTabResizeCbK);
0242

```



```

0243 else
0244 %-----
0245 % TopT/Bottom Tab
0246 %-----
0247 margins(3:4) = -(margins(1:2)+margins(3:4));
0248 switch type
0249 case 2 %lefttop
0250     [loop,sign] = deal(1:ntab,[1,0,1,0]);
0251 case 3 %righttop
0252     [loop,sign] = deal(ntab:-1:1,[ntab,0,0,0]);
0253 case 4 %lefttbottom
0254     [loop,sign] = deal(1:ntab,[1,0,1,1]);
0255 case 5 %rightbottom
0256     [loop,sign] = deal(ntab:-1:1,[ntab,0,0,1]);
0257 case 6 %centertop
0258     [loop,sign] = deal(1:ntab,[1,ntab,1,0]);
0259 case 7 %centerbottom
0260     [loop,sign] = deal(1:ntab,[1,ntab,1,1]);
0261 end
0262 addtitle = 2*[sign(3:4),-1,-1];
0263 if sign(3)
0264     ttextpos = [0.05,0.08,0.9,0.76];
0265 else
0266     ttextpos = [0.05,0.05,0.9,0.76];
0267 end
0268
0269 if ~exist('panelbordertype','var')
0270     panelbordertype = 'beveledout';
0271 end
0272
0273 for k=1:ntab
0274     title(k) = uipanel( ...
0275         'Parent',htab, ...
0276         'Units','pixels', ...
0277         'BackgroundColor',titlebackgroundcolor, ...
0278         'BorderType',panelbordertype, ...
0279         'ButtonDownFcn',{@TopBottomTabCbK,k}, ...
0280         'Visible','on');
0281     %titlewidth(k) = (length(string{k})+4)*1.2*charsz(3);
0282
0283     ttext(k) = uicontrol( ...
0284         'Parent',title(k), ...
0285         'Units','normalized', ...
0286         'Position',ttextpos, ...
0287         'ForegroundColor',foregroundcolor, ...
0288         'BackgroundColor',titlebackgroundcolor, ...
0289         'HorizontalAlignment',horizontalalignment, ...
0290         'Style','text', ...
0291         'String',string{k}, ...
0292         'Enable','inactive', ...
0293         'FontSize',fontsize, ...
0294         'HitTest','off', ...
0295         'Visible','on');
0296
0297     panel(k) = uipanel( ...
0298         'Parent',htab, ...
0299         'Units','pixels', ...
0300         'BackgroundColor',panelbackgroundcolor, ...
0301         'BorderType',panelbordertype, ...
0302         'Visible','off');
0303
0304     uimenu(cmenu, ...
0305         'Label',string{k}, ...
0306         'Callback',{@TopBottomTabCbK,k});
0307 end
0308
0309 if ~active, active = 1; end
0310 set(panel(active),'Visible','on');
0311
0312 inset(1) = uicontrol( ...
0313     'Parent',htab, ...
0314     'Units','pixels', ...
0315     'BackgroundColor',panelbackgroundcolor, ...
0316     'Style','text', ...
0317     'Visible','on');
0318 if ~isnan(titlehighlightcolor)
0319     inset(2) = uicontrol( ...
0320         'Parent',htab, ...
0321         'Units','pixels', ...
0322         'BackgroundColor',titlehighlightcolor, ...
0323         'Style','text', ...
0324         'Visible','on');
0325     inset(3) = uicontrol( ...
0326         'Parent',htab, ...
0327         'Units','pixels', ...
0328         'BackgroundColor',0.85*titlehighlightcolor, ...

```



```

0329         'Style','text',...
0330         'Visible','on');
0331     end
0332
0333     TopBottomTabResizeCbk(status,[]);
0334     set(status,'ResizeFcn',@TopBottomTabResizeCbk);
0335 end
0336
0337 setappdata(htab,'panels',panel);
0338 setappdata(htab,'status',status);
0339 if ~isempty(createfcn)
0340     createfcn(htab,[],panel,status);
0341 end
0342
0343 if narginout, hh = htab; end
0344
0345
0346 %-----
0347 % Inner Functions for Popup Tab
0348 %-----
0349 function PopupTabCbk(hobj,evdt,indx)
0350
0351     set(title,'Value',0);
0352     if indx>0
0353         indxvis = strcmp(get(panel(indx),'Visible'),'off');
0354         if indxvis
0355             set(panel(indx),'Visible','on');
0356         else
0357             set(panel(indx),'Visible','off');
0358         end
0359     end
0360     if isempty(strmatch('on',get(panel,'Visible')))
0361         set(status,'Visible','on');
0362     else
0363         set(status,'Visible','off');
0364     end
0365
0366     currval = get(slider,'Value');
0367     if currval<0
0368         set(slider,'Value',0);
0369         PopupTabSliderCbk(slider,evdt);
0370     end
0371     titlepos = get(title(1),'Position');
0372     addtolen = titlepos(2)+titlepos(4)+margins(4);
0373
0374     for k=1:ntab
0375         titlepos = get(title(k),'Position');
0376         titlepos(2) = addtolen-titlepos(4)-margins(4);
0377         set(title(k),'Position',titlepos);
0378         addtolen = titlepos(2);
0379         visible = strcmp(get(panel(k),'Visible'),'on');
0380         if visible
0381             panelpos = get(panel(k),'Position');
0382             addtolen = addtolen-panelpos(4);
0383             panelpos(2) = addtolen;
0384             set(panel(k),'Position',panelpos);
0385         end
0386     end
0387
0388     if visible
0389         botpos = get(panel(ntab),'Position');
0390     else
0391         botpos = get(title(ntab),'Position');
0392     end
0393
0394     visible = strcmp(get(slider,'Visible'),'on');
0395     addwidth = (-1)^(~visible);
0396     if botpos(2)*addwidth>0
0397         if visible
0398             set(slider,'Visible','off');
0399         else
0400             set(slider,'Visible','on');
0401         end
0402     else
0403         if visible
0404             addwidth = 0;
0405         else
0406             return;
0407         end
0408     end
0409
0410     addwidth = addwidth*sliderpos(3);
0411     for k=1:ntab
0412         titlepos = get(title(k),'Position');
0413         titlepos(3) = titlepos(3)+addwidth;
0414         set(title(k),'Position',titlepos);

```

```

0415     set(title(k), 'UserData', titlepos(2));
0416
0417     panelpos = get(panel(k), 'Position');
0418     panelpos(3) = panelpos(3)+addwidth;
0419     set(panel(k), 'Position', panelpos);
0420     set(panel(k), 'UserData', panelpos(2));
0421 end
0422
0423     toplen = botpos(2)-margins(2);
0424     if toplen>0
0425         set(slider, 'Value', 0);
0426         return;
0427     end
0428
0429     try
0430         if indxvis
0431             viewpos = get(panel(indx), 'Position');
0432             if viewpos(2)-viewpos(4)<0
0433                 if viewpos(4)+margins(4)<=tabpos(4);
0434                     currval = min(0, viewpos(2)-margins(4)/2);
0435                 else
0436                     viewpos = get(title(indx), 'Position');
0437                     currval = viewpos(2)+viewpos(4)+margins(4)/2 - tabpos(4);
0438                 end
0439             end
0440         end
0441     catch
0442     end
0443
0444     set(slider, 'Min', toplen);
0445     set(slider, 'Value', max(currval, toplen));
0446     set(slider, 'SliderStep', [0.3, 1]/max(1, -toplen/100));
0447     PopupTabSliderCbk(slider, evdt);
0448
0449 end
0450
0451
0452 %-----
0453 function PopupTabSliderCbk(hobj, evdt)
0454
0455     currval = get(hobj, 'Value');
0456     for k=1:ntab
0457         titlepos = get(title(k), 'Position');
0458         titlepos(2) = get(title(k), 'UserData')-currval;
0459         set(title(k), 'Position', titlepos);
0460         panelpos = get(panel(k), 'Position');
0461         panelpos(2) = get(panel(k), 'UserData')-currval;
0462         set(panel(k), 'Position', panelpos);
0463     end
0464
0465 end
0466
0467
0468 %-----
0469 function PopupTabEnableAllCbk(hobj, evdt, cstr)
0470
0471     set(panel, 'Visible', cstr);
0472     PopupTabCbk(hobj, evdt, 0);
0473
0474 end
0475
0476
0477 %-----
0478 function PopupTabResizeCbk(hobj, evdt)
0479
0480     oldpos = tabpos;
0481     units = get(htab, 'Units');
0482     set(htab, 'Units', 'pixels');
0483     tabpos = get(htab, 'Position');
0484     set(htab, 'Units', units);
0485
0486     addtopos = tabpos-oldpos;
0487     sliderpos([1, 2, 4]) = sliderpos([1, 2, 4])+addtopos([1, 2, 4]);
0488     set(slider, 'Position', sliderpos);
0489
0490     addtopos = [tabpos(3:4)-oldpos(3:4), 0, 0];
0491     currval = get(slider, 'Value');
0492     if currval<0
0493         set(slider, 'Value', 0);
0494         PopupTabSliderCbk(slider, evdt);
0495     end
0496
0497     for k=1:ntab
0498         titlepos = get(title(k), 'Position');
0499         set(title(k), 'Position', titlepos+addtopos);
0500     end

```

```

0501     PopupTabCbk(title(1),evdt,0);
0502
0503     visible = strcmp(get(slider,'Visible'),'on');
0504     if visible & currval<0
0505         set(slider,'Value',max(currval,get(slider,'Min')));
0506         PopupTabSliderCbk(slider,evdt);
0507     end
0508
0509 end
0510
0511 %-----
0512 % Inner Functions for Top/Bottom Tab
0513 %-----
0514 function TopBottomTabCbk(hobj,evdt,indx)
0515
0516     if active==indx, return; end
0517
0518     % deactivate tab(active)
0519     n = active;
0520     titlepos = get(title(n),'Position');
0521     titlepos = titlepos+addtitle.*[n==sign(1),1,n==sign(1),1];
0522     titlepos(3) = titlepos(3)+addtitle(3)*(n==sign(2));
0523     set(title(n),'Position',titlepos,'BackgroundColor',titlebackgroundcolor);
0524     set(ttext(n),'FontWeight','normal','BackgroundColor',titlebackgroundcolor);
0525     set(panel(n),'Visible','off');
0526
0527     % activate tab(indx)
0528     active = indx;
0529     titlepos = get(title(indx),'Position');
0530     titlepos = titlepos-addtitle.*[indx==sign(1),1,indx==sign(1),1];
0531     titlepos(3) = titlepos(3)-addtitle(3)*(indx==sign(2));
0532
0533     set(title(indx),'Position',titlepos,'BackgroundColor',panelbackgroundcolor);
0534     set(ttext(indx),'FontWeight','bold','BackgroundColor',panelbackgroundcolor);
0535     insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
0536     set(inset(1),'Position',insetpos);
0537     try
0538         insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
0539         set(inset(2),'Position',insetpos);
0540         set(inset(3),'Position',insetpos+[2,1-2*sign(4),-4,0]);
0541     catch
0542     end
0543     set(panel(indx),'Visible','on');
0544
0545     if isappdata(htab,'SelectionChangeFcn')
0546         SelectionChangeFcn = getappdata(htab,'SelectionChangeFcn');
0547         eventdata = struct('EventName','SelectionChanged','...
0548             'OldValue',n,'NewValue',indx,'Panels',panel,'Status',status);
0549         if iscell(SelectionChangeFcn)
0550             SelectionChangeFcn{1}(hobj,eventdata,SelectionChangeFcn{2:end});
0551         else
0552             SelectionChangeFcn(hobj,eventdata);
0553         end
0554     end
0555 end
0556
0557 end
0558
0559 %-----
0560 function TopBottomTabResizeCbk(hobj,evdt)
0561
0562     units = get(htab,'Units');
0563     set(htab,'Units','pixels');
0564     tabpos = get(htab,'Position');
0565     set(htab,'Units',units);
0566
0567     titlepos = [0,0,0,titleheight];
0568     if sign(4)
0569         panelpos = [0,titleheight+1,tabpos(3),tabpos(4)-titleheight]+margins;
0570         titlepos(2) = panelpos(2)-titlepos(4);
0571     else
0572         panelpos = [0,0,tabpos(3),tabpos(4)-titleheight]+margins;
0573         titlepos(2) = panelpos(2)+panelpos(4);
0574     end
0575     set(panel,'Position',panelpos);
0576
0577     n = sum(cellfun('length',string))+2*ntab;
0578     if type<6
0579         if (n+2*ntab)*1.2*charsz(3)+2>panelpos(3)
0580             [loop,sign(1:3)] = deal(1:ntab,[1,ntab,1]);
0581         else
0582             if type==2 | type==4
0583                 [loop,sign(1:3)] = deal(1:ntab,[1,0,1]);
0584             else
0585                 [loop,sign(1:3)] = deal(ntab:-1:1,[ntab,0,0]);
0586             end
0587         end
0588     end

```

```

0587         end
0588     end
0589     addtitle = 2*[sign(3:4), -1, -1];
0590 end
0591 titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
0592 for k=loop
0593     if sign(2)
0594         titlepos(3) = panelpos(3)*(length(string{k})+2)/n;
0595         addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
0596     else
0597         titlepos(3) = (length(string{k})+4)*1.2*charsz(3);
0598         addtopos = addtitle.*[k==sign(1),1,k==sign(1),1];
0599     end
0600     titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
0601     set(title(k), 'Position', titlepos+addtopos);
0602     titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
0603 end
0604
0605 statuspos = titlepos;
0606 statuspos(1) = sign(3)*titlepos(1)+2;
0607 statuspos(3) = titlepos(1)+sign(3)*(tabpos(3)-2*titlepos(1))-4;
0608
0609 titlepos = get(title(active), 'Position');
0610 titlepos = titlepos-addtitle.*[active==sign(1),1,active==sign(1),1];
0611 titlepos(3) = titlepos(3)-addtitle(3)*(active==sign(2));
0612 set(title(active), 'Position', titlepos, 'BackgroundColor', panelbackgroundcolor);
0613 set(ttext(active), 'FontWeight', 'bold', 'BackgroundColor', panelbackgroundcolor);
0614
0615 insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
0616 set(inset(1), 'Position', insetpos);
0617 try
0618     insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
0619     set(inset(2), 'Position', insetpos);
0620     set(inset(3), 'Position', insetpos+[2,1-2*sign(4),-4,0]);
0621 catch
0622 end
0623 try
0624     % This causes the above statements being excuted twice for each resizing.
0625     set(status, 'Position', statuspos, 'Visible', 'on');
0626 catch
0627     set(status, 'Visible', 'off');
0628 end
0629
0630 end
0631
0632 end

```

Contents

PURPOSE ^

%% path: m/gui/ltpdaRepoGUI/callbacks
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/ltpdaRepoGUI/callbacks %%%%%%%%%%  
  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/buildConditions">m/gui/ltpdaRepoGUI/callbacks/buildConditions</a>  
- Callback to build the query conditions panel.  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/buildquery">m/gui/ltpdaRepoGUI/callbacks/buildquery</a>  
- a query from the settings on the query panel  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_classMethodList">m/gui/ltpdaRepoGUI/callbacks/cb_classMethodList</a>  
- Get a list of methods for a class and fill the list  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_condPlusBtn">m/gui/ltpdaRepoGUI/callbacks/cb_condPlusBtn</a>  
- Add a condition to the query panel  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_condSubtractBtn">m/gui/ltpdaRepoGUI/callbacks/cb_condSubtractBtn</a>  
- Remove a condition from the query panel  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_connectBtn">m/gui/ltpdaRepoGUI/callbacks/cb_connectBtn</a>  
- Callback to connect to server  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_dbSelect">m/gui/ltpdaRepoGUI/callbacks/cb_dbSelect</a>  
- Callback for when user selects a database  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_executeQuery">m/gui/ltpdaRepoGUI/callbacks/cb_executeQuery</a>  
- the query built on the query panel  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_fieldSelect">m/gui/ltpdaRepoGUI/callbacks/cb_fieldSelect</a>  
- Callback executed when a field is selected  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_getDBs">m/gui/ltpdaRepoGUI/callbacks/cb_getDBs</a>  
- Callback executed when the user clicks the 'Get DBs' button  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_importBtn">m/gui/ltpdaRepoGUI/callbacks/cb_importBtn</a>  
- Callback executed when the user clicks on the import button.  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_mainfigClose">m/gui/ltpdaRepoGUI/callbacks/cb_mainfigClose</a>  
- Callback executed when the main figure closes  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsList">m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsList</a>  
- Callback to refresh the list of objects.  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsListQL">m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsListQL</a>  
- Callback to refresh the list of objects.  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_retrieveBtn">m/gui/ltpdaRepoGUI/callbacks/cb_retrieveBtn</a>  
- Callback executed when the user clicks on the retrieve button.  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_saveBtn">m/gui/ltpdaRepoGUI/callbacks/cb_saveBtn</a>  
- Callback executed when the user clicks on the save button.  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_serverSelect">m/gui/ltpdaRepoGUI/callbacks/cb_serverSelect</a>  
- Callback executed when the user selects a server.  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_submitBtn">m/gui/ltpdaRepoGUI/callbacks/cb_submitBtn</a>  
- Callback executed when the user clicks on the 'Submit' button  
    <a href="matlab:help  
m/gui/ltpdaRepoGUI/callbacks/cb_tableSelect">m/gui/ltpdaRepoGUI/callbacks/cb_tableSelect</a>  
- Callback executed when a table selection is made  
    <a href="matlab:help
```

```
m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsList">m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsList
- Callback to fill the list of workspace objects
  <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsListQL">m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsListQL
- Callback to fill the list of workspace objects
  <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/fillClassMethodsList">m/gui/ltpdaRepoGUI/callbacks/fillClassMethodsList
- Callback to fill the list of class methods
  <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/getFields">m/gui/ltpdaRepoGUI/callbacks/getFields</a>
- a list of database fields
  <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/getTables">m/gui/ltpdaRepoGUI/callbacks/getTables</a>
- a list of database tables
  <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/getWorkspaceObjs">m/gui/ltpdaRepoGUI/callbacks/getWorkspaceObjs</a>
- a list of LTPDA objects in the MATLAB workspace
  <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/setWorkspaceObjsList">m/gui/ltpdaRepoGUI/callbacks/setWorkspaceObjsList
- Fill the workspace object list
  <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/setWorkspaceObjsListQL">m/gui/ltpdaRepoGUI/callbacks/setWorkspaceObjsListQL
- Fill the workspace object list
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > buildConditions.m

buildConditions

PURPOSE [^]

Callback to build the query conditions panel.

SYNOPSIS [^]

function buildConditions(varargin)

DESCRIPTION [^]

Callback to build the query conditions panel.
 \$Id: buildConditions.m,v 1.5 2008/02/23 14:24:55 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [buildquery](#) Build a query from the settings on the query panel
-  [getFields](#) Get a list of database fields
-  [getTables](#) Get a list of database tables

This function is called by:

-  [cb_condPlusBtn](#) Add a condition to the query panel
-  [cb_condSubtractBtn](#) Remove a condition from the query panel
-  [cb_connectBtn](#) Callback to connect to server

SOURCE CODE [^]

```
0001 function buildConditions(varargin)
0002
0003 % Callback to build the query conditions panel.
0004 %
0005 % $Id: buildConditions.m,v 1.5 2008/02/23 14:24:55 hewitson Exp $
0006 %
0007
0008 ALLHEIGHT = 25;
0009 fontsize = 10;
0010
0011 panel = varargin{1};
0012 Ppos = get(panel, 'Position');
0013 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0014 Nc = getappdata(mainfig, 'Nconditions');
0015 conn = getappdata(mainfig, 'connection');
0016 Gproperties = getappdata(mainfig, 'Gproperties');
0017
0018 % before deleting, record the table selection
0019 tableSelected = getappdata(mainfig, 'tableSelected');
0020 % record field selection
0021 fieldList = findobj('Tag', 'fieldsList');
0022 fselection = get(fieldList, 'Value');
0023 % record order by
0024 fieldOrder = findobj('Tag', 'fieldOrderBy');
0025 forder = get(fieldOrder, 'Value');
0026 % record sort direction
0027 sortDir = findobj('Tag', 'sortDir');
0028 sort = get(sortDir, 'Value');
0029 % record current conditions
0030 conds = cell(Nc,4);
0031 for k=1:Nc
```

```

0032 pmh = findobj('Tag', sprintf('c%02dField', k));
0033 conds{k,1} = get(pmh, 'Value');
0034 pmh = findobj('Tag', sprintf('c%02dCondition', k));
0035 conds{k,2} = get(pmh, 'Value');
0036 pmh = findobj('Tag', sprintf('c%02dValue', k));
0037 conds{k,3} = get(pmh, 'String');
0038 if k<Nc
0039     pmh = findobj('Tag', sprintf('c%02dCompound', k));
0040     conds{k,4} = get(pmh, 'Value');
0041 end
0042 end
0043
0044 delete(get(panel, 'Children'))
0045 drawnow
0046 %-----
0047 %-- Draw Table selection
0048
0049 % get tables
0050 tables = getTables(conn);
0051 Sleft = 10;
0052 Sheight = ALLHEIGHT;
0053 Sbottom = Ppos(2) + Ppos(4) - 20 - Sheight;
0054 Swidth = 50;
0055
0056 sth = uicontrol(panel, 'Style', 'text', ...
0057     'String', 'From', ...
0058     'FontSize', fontsize, ...
0059     'BackgroundColor', Gproperties.Gcol, ...
0060     'HorizontalAlignment', 'left', ...
0061     'Position', [Sleft Sbottom Swidth Sheight]);
0062
0063 Sleft = Sleft + Swidth + 10;
0064 Swidth = 150;
0065 pmh = uicontrol(panel, 'Style', 'popupmenu', ...
0066     'String', tables, ...
0067     'FontSize', fontsize, ...
0068     'BackgroundColor', Gproperties.Gcol, ...
0069     'Value', 1, ...
0070     'Position', [Sleft Sbottom Swidth Sheight], ...
0071     'Tag', 'tableSelect');
0072 if ~isempty(tableSelected)
0073     set(pmh, 'Value', tableSelected);
0074 end
0075 set(pmh, 'Callback', {'cb_tableSelect', pmh, mainfig});
0076
0077 %-----
0078 %-- Draw Fields Selection list
0079
0080 % get fields
0081 fields = getFields(conn);
0082 pos = get(pmh, 'Position');
0083 Sleft = 10;
0084 Sheight = ALLHEIGHT;
0085 Sbottom = pos(2) + pos(4) - 40 - Sheight;
0086 Swidth = 50;
0087
0088 sth = uicontrol(panel, 'Style', 'text', ...
0089     'String', 'Select', ...
0090     'FontSize', fontsize, ...
0091     'BackgroundColor', Gproperties.Gcol, ...
0092     'HorizontalAlignment', 'left', ...
0093     'Position', [Sleft Sbottom Swidth Sheight]);
0094
0095 Sleft = Sleft + Swidth + 10;
0096 Swidth = 150;
0097 Sheight = 200;
0098 Sbottom = pos(2) + pos(4) - 40 - Sheight;
0099 lbh = uicontrol(panel, 'Style', 'listbox', ...
0100     'String', fields, ...
0101     'Value', 1, ...
0102     'BackgroundColor', 'w', ...
0103     'FontSize', fontsize, ...
0104     'Max', 1000, ...
0105     'Position', [Sleft Sbottom Swidth Sheight], ...
0106     'Tag', 'fieldsList');
0107
0108 set(lbh, 'Callback', {'cb_fieldSelect', pmh, mainfig});
0109 if ~isempty(fselection)
0110     set(lbh, 'Value', fselection);
0111 end
0112
0113 %-----
0114 %-- Query display
0115 listpos = get(lbh, 'Position');
0116 Sleft = 10;
0117 Sheight = 40;

```



```

0118 Sbottom = Ppos(2) + 60; %listpos(2) - 40 - Sheight;
0119 Swidth = 60;
0120
0121 sth = uicontrol(panel,'Style','text',...
0122     'String','Query:',...
0123     'FontSize', fontsize,...
0124     'Max', 5,...
0125     'BackgroundColor', Gproperties.Gcol,...
0126     'HorizontalAlignment','left',...
0127     'Position', [Sleft Sbottom Swidth Sheight]);
0128
0129 Sleft = 10 + Swidth + 20;
0130 Swidth = 650;
0131 Sheight = 100;
0132 Sbottom = Ppos(2) + 60; %listpos(2) - 40 - Sheight;
0133 sth = uicontrol(panel,'Style','edit',...
0134     'String','',...
0135     'FontSize', fontsize,...
0136     'Max', 5,...
0137     'BackgroundColor', 'w',...
0138     'HorizontalAlignment','left',...
0139     'Position', [Sleft Sbottom Swidth Sheight],...
0140     'Tag', 'queryDisplayTxt');
0141
0142
0143
0144 %-----
0145 %-- Draw conditions
0146
0147 conditions = {'=', '<>', '<=', '>=', '<', '>', 'like', 'not like', 'is'};
0148 compounds = {'AND', 'OR', 'XOR'};
0149
0150 pos = get(lbh, 'Position');
0151 Sleft = pos(1) + pos(3) + 20;
0152 Sheight = ALLHEIGHT;
0153 Sbottom = Ppos(2) + Ppos(4) - 25 - Sheight;
0154 Swidth = 50;
0155
0156 sth = uicontrol(panel,'Style','text',...
0157     'String','Where',...
0158     'FontSize', fontsize,...
0159     'BackgroundColor', Gproperties.Gcol,...
0160     'HorizontalAlignment','left',...
0161     'Position', [Sleft Sbottom Swidth Sheight]);
0162
0163
0164 % pos = get(sth, 'Position');
0165 Cheight = ALLHEIGHT;
0166 STEP = 5;
0167 for k=1:Nc
0168     Cleft = pos(1) + pos(3) + 10 + STEP;
0169     Cbottom = pos(2) + pos(4) - 25 - (k-1)*(Cheight+10);
0170     Cwidth = 150;
0171     % Field Dropdown
0172     pmh = uicontrol(panel,'Style','popupmenu',...
0173         'String', fields,...
0174         'FontSize', fontsize,...
0175         'BackgroundColor', 'w',...
0176         'Value', 1, 'Position', [Cleft Cbottom Cwidth Cheight],...
0177         'Tag', sprintf('c%02dField', k));
0178     set(pmh, 'Callback', 'buildquery');
0179     if ~isempty(conds{k,1})
0180         set(pmh, 'Value', conds{k,1});
0181     end
0182     % Gproperties.Gcol,
0183     % Condition Dropdown
0184     Cleft = Cleft + Cwidth + STEP;
0185     Cwidth = 100;
0186     pmh = uicontrol(panel,'Style','popupmenu',...
0187         'String', conditions,...
0188         'FontSize', fontsize,...
0189         'BackgroundColor', 'w',...
0190         'Value', 1, 'Position', [Cleft Cbottom Cwidth Cheight],...
0191         'Tag', sprintf('c%02dCondition', k));
0192     set(pmh, 'Callback', 'buildquery');
0193     if ~isempty(conds{k,2})
0194         set(pmh, 'Value', conds{k,2});
0195     end
0196
0197     % Value txt box
0198     Cleft = Cleft + Cwidth + STEP;
0199     Cwidth = 150;
0200     sth = uicontrol(panel,'Style','edit',...
0201         'String','',...
0202         'FontSize', fontsize,...
0203         'BackgroundColor', 'w',...

```

```

0204     'HorizontalAlignment', 'left',...
0205     'Position',[Cleft Cbottom Cwidth Cheight],...
0206     'Tag', sprintf('c%02dValue', k));
0207 set(sth, 'Callback', 'buildquery');
0208 if ~isempty(conds{k,3})
0209     set(sth, 'String', conds{k,3});
0210 end
0211
0212 % Add compound statement
0213 if k<Nc
0214     Cleft = Cleft + Cwidth + STEP;
0215     Cwidth = 80;
0216     pmh = uicontrol(panel, 'Style', 'popupmenu',...
0217         'String', compounds,...
0218         'FontSize', 10,...
0219         'BackgroundColor', Gproperties.Gcol,...
0220         'Value', 1, 'Position',[Cleft Cbottom Cwidth Cheight],...
0221         'Tag', sprintf('c%02dCompound', k));
0222     set(pmh, 'Callback', 'buildquery');
0223     if ~isempty(conds{k,4})
0224         set(pmh, 'Value', conds{k,4});
0225     end
0226 end
0227
0228 end
0229
0230 %-----
0231 %-- add +/- buttons
0232 k = Nc+1;
0233 Bbottom = pos(2) + pos(4) - 25 - (k-1)*(Cheight+10);
0234 Bleft = pos(1) + pos(3) + 20;
0235 Bheight = ALLHEIGHT;
0236 Bwidth = 20;
0237 pbh = uicontrol(panel, 'Style', 'pushbutton', 'FontSize', fontsize, 'String', '+',...
0238     'Position',[Bleft Bbottom Bwidth Bheight ],...
0239     'Tag', 'condPlusBtn');
0240 set(pbh, 'Callback', {'cb_condPlusBtn', pbh, panel, mainfig});
0241 Bleft = pos(1) + pos(3) + 20 + 30;
0242 pbh = uicontrol(panel, 'Style', 'pushbutton', 'FontSize', fontsize, 'String', '-',...
0243     'Position',[Bleft Bbottom Bwidth Bheight ],...
0244     'Tag', 'condSubtractBtn');
0245 set(pbh, 'Callback', {'cb_condSubtractBtn', pbh, panel, mainfig});
0246
0247
0248 %-----
0249 %-- order by
0250 k = Nc + 2;
0251 Sleft = pos(1) + pos(3) + 20;
0252 Sheight = ALLHEIGHT;
0253 Swidth = 60;
0254 Sbottom = pos(2) + pos(4) - 25 - (k-1)*(Cheight+10);
0255 sth = uicontrol(panel, 'Style', 'text',...
0256     'String', 'Order by',...
0257     'FontSize', fontsize,...
0258     'BackgroundColor', Gproperties.Gcol,...
0259     'HorizontalAlignment', 'left',...
0260     'Position',[Sleft Sbottom Swidth Sheight]);
0261 % Gproperties.Gcol
0262 % Field Dropdown
0263 Sleft = pos(1) + pos(3) + 20 + Swidth + 20;
0264 Swidth = 100;
0265 pmh = uicontrol(panel, 'Style', 'popupmenu',...
0266     'String', fields,...
0267     'FontSize', fontsize,...
0268     'BackgroundColor', Gproperties.Gcol,...
0269     'Value', 1, 'Position',[Sleft Sbottom Swidth Sheight],...
0270     'Tag', 'fieldOrderBy');
0271 set(pmh, 'Callback', 'buildquery');
0272 if ~isempty(forder)
0273     set(pmh, 'Value', forder);
0274 end
0275
0276 % Desc Dropdown
0277 Sleft = Sleft + 20 + Swidth;
0278 Swidth = 100;
0279 pmh = uicontrol(panel, 'Style', 'popupmenu',...
0280     'String', {'DESC', 'ASC'},...
0281     'FontSize', fontsize,...
0282     'BackgroundColor', Gproperties.Gcol,...
0283     'Value', 1, 'Position',[Sleft Sbottom Swidth Sheight],...
0284     'Tag', 'sortDir');
0285 set(pmh, 'Callback', 'buildquery');
0286 if ~isempty(sort)
0287     set(pmh, 'Value', sort);
0288 end
0289

```

```
0290
0291 %-----
0292 %-- Run query
0293
0294 k = Nc + 3;
0295 Sleft = Ppos(1) + 10;
0296 Sheight = ALLHEIGHT;
0297 Swidth = 100;
0298 Sbottom = Ppos(2) + 10;
0299 pbh = uicontrol(panel, 'Style', 'pushbutton', 'String', 'Execute query', ...
0300                  'Position', [Sleft Sbottom Swidth Sheight ], ...
0301                  'Tag', 'refreshListBtn');
0302 set(pbh, 'Callback', {'cb_executeQuery', pbh, mainfig});
0303
0304
0305 % Build query
0306 buildquery()
0307
0308 drawnow
0309
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > buildquery.m

buildquery

PURPOSE [^]

Build a query from the settings on the query panel

SYNOPSIS [^]

function buildquery()

DESCRIPTION [^]

```
Build a query from the settings on the query panel
M Hewitson
$Id: buildquery.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

-  [buildConditions](#) Callback to build the query conditions panel.
-  [cb_fieldSelect](#) Callback executed when a field is selected
-  [cb_tableSelect](#) Callback executed when a table selection is made

SOURCE CODE [^]

```
0001 function buildquery()
0002
0003 % Build a query from the settings on the query panel
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: buildquery.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0008 %
0009
0010 mainfig      = findobj('Tag', 'LTPDAREpomainfig');
0011
0012 % Get number of conditions
0013 Nc = getappdata(mainfig, 'Nconditions');
0014
0015 % Select fields
0016 fieldList = findobj('Tag', 'fieldsList');
0017 fieldsStr = get(fieldList, 'String');
0018 vals      = get(fieldList, 'Value');
0019 fields    = fieldsStr(vals);
0020
0021 % Get table
0022 tableSel = findobj('Tag', 'tableSelect');
0023 tbls     = get(tableSel, 'String');
0024 tbl      = tbls{get(tableSel, 'Value')};
0025
0026 % Start the query
0027 q = 'SELECT ';
0028 for j=1:length(fields)
0029     q = [q fields{j} ','];
0030 end
0031 q = [q(1:end-1) ' FROM ' tbl];
0032
0033
0034 for k=1:Nc
0035
0036     if k==1
```

```

0037     q = [q ' WHERE '];
0038 end
0039
0040 % get field
0041 h = findobj('Tag', sprintf('c%02dField', k));
0042 fields = get(h, 'String');
0043 field = fields{get(h, 'Value')};
0044
0045 q = [q field];
0046
0047 % get condition
0048 h = findobj('Tag', sprintf('c%02dCondition', k));
0049 conds = get(h, 'String');
0050 cond = conds{get(h, 'Value')};
0051
0052 q = [q ' ' cond ' '];
0053
0054 % get value
0055 h = findobj('Tag', sprintf('c%02dValue', k));
0056 val = get(h, 'String');
0057 q = [q ' ' val ' '];
0058
0059 % Get compound
0060 if k<Nc
0061     h = findobj('Tag', sprintf('c%02dCompound', k));
0062     comps = get(h, 'String');
0063     comp = comps{get(h, 'Value')};
0064
0065     q = [q ' ' comp ' '];
0066
0067 end
0068 end
0069
0070 end
0071 end
0072
0073 % Order by
0074 h = findobj('Tag', 'fieldOrderBy');
0075 fields = get(h, 'String');
0076 field = fields{get(h, 'Value')};
0077
0078 q = [q ' ORDER BY ' field];
0079
0080 h = findobj('Tag', 'sortDir');
0081 dirs = get(h, 'String');
0082 dir = dirs{get(h, 'Value')};
0083
0084 q = [q ' ' dir ''];
0085
0086 % Set query string on GUI
0087 h = findobj('Tag', 'queryDisplayTxt');
0088 set(h, 'String', q);
0089
0090 drawnow

```

cb_classMethodList

PURPOSE ^

Get a list of methods for a class and fill the list

SYNOPSIS ^

function cb_classMethodList(varargin)

DESCRIPTION ^

Get a list of methods for a class and fill the list
M Hewitson
\$Id: cb_classMethodList.m,v 1.1 2008/01/22 21:10:23 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_classMethodList(varargin)
0002
0003 % Get a list of methods for a class and fill the list
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_classMethodList.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
0008 %
0009
0010 myh      = varargin{1};
0011 mainfig = varargin{4};
0012
0013 id = findobj('Tag', 'methodInfoPanel');
0014
0015 % get selected method
0016 id = findobj('Tag', 'classMethodList');
0017 ostr = get(id, 'String');
0018 oidx = get(id, 'Value');
0019
0020 objStr = ostr(oidx);
0021
0022 id = findobj('Tag', 'methodInfoPanel');
0023
0024 if length(oidx) == 1
0025     s = help(char(objStr));
0026     set(id, 'String', s);
0027 end
0028
```


cb_condPlusBtn

PURPOSE [^]

Add a condition to the query panel

SYNOPSIS [^]

function cb_condPlusBtn(varargin)

DESCRIPTION [^]

Add a condition to the query panel
M Hewitson
\$Id: cb_condPlusBtn.m,v 1.3 2008/02/23 13:44:08 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [buildConditions](#) Callback to build the query conditions panel.

This function is called by:

SOURCE CODE [^]

```
0001 function cb_condPlusBtn(varargin)
0002
0003 % Add a condition to the query panel
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_condPlusBtn.m,v 1.3 2008/02/23 13:44:08 hewitson Exp $
0008 %
0009
0010 myh = varargin{1};
0011
0012 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0013 Nc = getappdata(mainfig, 'Nconditions');
0014 if Nc >= 8
0015     Nc = 7;
0016     warning('!!! Can''t have more than 8 conditions.');
```


cb_condSubtractBtn

PURPOSE ^

Remove a condition from the query panel

SYNOPSIS ^

function cb_condSubtractBtn(varargin)

DESCRIPTION ^

Remove a condition from the query panel
M Hewitson
\$Id: cb_condSubtractBtn.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

 [buildConditions](#) Callback to build the query conditions panel.

This function is called by:

SOURCE CODE ^

```
0001 function cb_condSubtractBtn(varargin)
0002
0003 % Remove a condition from the query panel
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_condSubtractBtn.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0008 %
0009
0010 myh = varargin{1};
0011
0012 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0013 Nc = getappdata(mainfig, 'Nconditions');
0014 if Nc < 1
0015     Nc = 1;
0016 end
0017 setappdata(mainfig, 'Nconditions', Nc - 1);
0018
0019 buildConditions(get(myh, 'Parent'));
0020
0021
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_connectBtn.m](#)

cb_connectBtn

PURPOSE ^

Callback to connect to server

SYNOPSIS ^

```
function cb_connectBtn(varargin)
```

DESCRIPTION ^

```
Callback to connect to server
M Hewitson
$Id: cb_connectBtn.m,v 1.5 2008/08/11 15:00:06 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [buildConditions](#) Callback to build the query conditions panel.

This function is called by:

SOURCE CODE ^

```
0001 function cb_connectBtn(varargin)
0002
0003 % Callback to connect to server
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_connectBtn.m,v 1.5 2008/08/11 15:00:06 hewitson Exp $
0008 %
0009
0010 myh      = varargin{1};
0011 mainfig = varargin{4};
0012 status  = findobj('Tag', 'statusTxt');
0013 panels  = getappdata(mainfig, 'panels');
0014
0015 % Get conn
0016 conn = getappdata(mainfig, 'connection');
0017
0018 if isempty(conn)
0019
0020     % get hostname
0021     hostname = get(findobj('Tag', 'serverTxt'), 'String');
0022
0023     % get the database
0024     db = get(findobj('Tag', 'databaseTxt'), 'String');
0025
0026     % check for username and password
0027     dbuser = getappdata(mainfig, 'dbuser');
0028     dbpass = getappdata(mainfig, 'dbpass');
0029
0030     % connect
0031     if ~isempty(dbpass) && ~isempty(dbuser)
0032         conn = utils.mysql.connect(hostname, db, dbuser, dbpass);
0033     else
0034         [conn, dbpass] = utils.mysql.connect(hostname, db);
0035     end
0036     if isa(conn, 'database')
0037         setappdata(mainfig, 'connection', conn);
0038         setappdata(mainfig, 'username', conn.Username);
0039         set(myh, 'String', 'disconnect');
```

```
0040     set(status, 'String', sprintf('connected to %s on %s as %s', db, hostname,  
conn.Username));  
0041  
0042     % Set tables on query panel  
0043     buildConditions(panels(3));  
0044  
0045     % store the connection  
0046     setappdata(mainfig, 'dbuser', conn.Username);  
0047     setappdata(mainfig, 'dbpass', dbpass);  
0048 else  
0049     set(status, 'String', 'not connected');  
0050     error('Invalid connection.');
```

```
0051 end  
0052 else  
0053     % disconnect  
0054     close(conn);  
0055     conn = [];  
0056     disp('* disconnected.');
```

```
0057     setappdata(mainfig, 'connection', conn);  
0058     setappdata(mainfig, 'username', '');  
0059     set(myh, 'String', 'connect');  
0060     set(status, 'String', 'not connected');  
0061     setappdata(mainfig, 'dbuser', '');  
0062     setappdata(mainfig, 'dbpass', '');  
0063 end  
0064
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > cb_dbSelect.m

cb_dbSelect

PURPOSE ^

Callback for when user selects a database

SYNOPSIS ^

```
function cb_dbSelect(varargin)
```

DESCRIPTION ^

```
Callback for when user selects a database
M Hewitson
$Id: cb_dbSelect.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_dbSelect(varargin)
0002
0003 % Callback for when user selects a database
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_dbSelect.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0008 %
0009
0010 myh      = varargin{1};
0011 mainfig  = findobj('Tag', 'LTPDAREpomainfig');
0012
0013 % Get selected DB
0014 dbh = findobj('Tag', 'dbSelect');
0015 dbs = get(dbh, 'String');
0016 db  = dbs{get(dbh, 'Value')};
0017
0018 % set selected DB
0019 dbh = findobj('Tag', 'databaseTxt');
0020 set(dbh, 'String', db);
0021
0022
0023 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_executeQuery.m](#)

cb_executeQuery

PURPOSE ^

Execute the query built on the query panel

SYNOPSIS ^

```
function cb_executeQuery(varargin)
```

DESCRIPTION ^

```
Execute the query built on the query panel
M Hewitson
$Id: cb_executeQuery.m,v 1.4 2008/02/22 09:25:18 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_executeQuery(varargin)
0002
0003 % Execute the query built on the query panel
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_executeQuery.m,v 1.4 2008/02/22 09:25:18 hewitson Exp $
0008 %
0009
0010 % Get connection
0011 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0012 conn = getappdata(mainfig, 'connection');
0013 if isempty(conn)
0014     error('### Please connect to a database first.');
```

```
0015 end
0016
0017 % Get Selected fields
0018 fieldList = findobj('Tag', 'fieldsList');
0019 fieldsStr = get(fieldList, 'String');
0020 vals = get(fieldList, 'Value');
0021 fields = fieldsStr(vals);
0022
0023
0024 % Get query
0025 h = findobj('Tag', 'queryDisplayTxt');
0026 q = get(h, 'String');
```

```
0027 try
0028     curs = exec(conn, q);
0029     curs = fetch(curs);
0030     results = curs.Data;
0031     close(curs);
0032 catch
0033     warning('!!! Unable to execute query %s', q)
0034     warning('!!! Server returned: %s', curs.Message);
0035     results = {};
0036 end
0037
0038
0039 if ~isempty(results)
0040     if strcmp(results{1}, 'No Data')
0041         errorlg('Query returned no results.', 'Query Empty');
```

```
0042     return
```



```
0043     else
0044         whos('results')
0045         whos('fields')
0046         sqlResultsGUI(results, fields, q);
0047     end
0048 else
0049     error('### Unable to execute query');
0050 end
0051
```

cb_fieldSelect

PURPOSE ^

Callback executed when a field is selected

SYNOPSIS ^

function cb_fieldSelect(varargin)

DESCRIPTION ^

Callback executed when a field is selected
M Hewitson
\$Id: cb_fieldSelect.m,v 1.3 2008/02/07 19:27:37 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

 [buildquery](#) Build a query from the settings on the query panel

This function is called by:

SOURCE CODE ^

```
0001 function cb_fieldSelect(varargin)
0002
0003 % Callback executed when a field is selected
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_fieldSelect.m,v 1.3 2008/02/07 19:27:37 hewitson Exp $
0008 %
0009 %
0010 %---- Set the order by field to the first in this list
0011
0012 % Get selected fields
0013 fieldList = findobj('Tag', 'fieldsList');
0014 fieldsStr = get(fieldList, 'String');
0015 vals      = get(fieldList, 'Value');
0016 fields    = fieldsStr(vals);
0017
0018 orderBy = findobj('Tag', 'fieldOrderBy');
0019 set(orderBy, 'Value', 1);
0020 set(orderBy, 'String', fields);
0021
0022
0023 % Build query
0024 buildquery();
0025
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_getDBs.m](#)

cb_getDBs

PURPOSE ^

Callback executed when the user clicks the 'Get DBs' button

SYNOPSIS ^

```
function cb_getDBs(varargin)
```

DESCRIPTION ^

```
Callback executed when the user clicks the 'Get DBs' button
M Hewitson
$Id: cb_getDBs.m,v 1.6 2008/03/11 14:33:51 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_getDBs(varargin)
0002
0003 % Callback executed when the user clicks the 'Get DBs' button
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_getDBs.m,v 1.6 2008/03/11 14:33:51 hewitson Exp $
0008 %
0009
0010 myh      = varargin{1};
0011 mainfig = varargin{4};
0012
0013 tables = {'collections', 'objmeta', 'objs', 'transactions', 'users'};
0014
0015 % Get conn
0016 conn = getappdata(mainfig, 'connection');
0017 % Assume I don't need to open the connection
0018 wasEmpty = 0;
0019
0020 % Open connection, if necessary
0021 if isempty(conn)
0022
0023     % we opened the connection
0024     wasEmpty = 1;
0025
0026     % have we a stored username and password
0027     dbuser = getappdata(mainfig, 'dbuser');
0028     dbpass = getappdata(mainfig, 'dbpass');
0029
0030     % get hostname
0031     hostname = get(findobj('Tag','serverTxt'), 'String');
0032
0033     % connect
0034     if ~isempty(dbuser) && ~isempty(dbpass)
0035         conn = mysql_connect(hostname, 'test', dbuser, dbpass);
0036     else
0037         [conn, dbpass] = mysql_connect(hostname);
0038     end
0039     if ~isa(conn, 'database')
0040         error('# couldn't connect to server');
0041     end
0042
```

```

0043 % store the connection
0044 setappdata(mainfig, 'dbuser', conn.Username);
0045 setappdata(mainfig, 'dbpass', dbpass);
0046
0047 %
0048 % % update the connect button and status string
0049 % status = findobj('Tag', 'statusTxt');
0050 % connBtn = findobj('Tag', 'connectBtn');
0051 % set(connBtn, 'String', 'disconnect');
0052 % set(status, 'String', sprintf('connected to %s on %s as %s', db, hostname,
conn.Username));
0053
0054 end
0055
0056 % get the databases
0057 q = 'show databases';
0058 curs = exec(conn, q);
0059 curs = fetch(curs);
0060
0061 ltpda_dbs = [];
0062 % Now get the LTPDA databases
0063 for j=1:length(curs.Data)
0064     db = curs.Data{j};
0065
0066     if ~strcmp(db, 'information_schema')
0067
0068         q = sprintf('show tables from %s', db);
0069         c = exec(conn, q);
0070         c = fetch(c);
0071         match = 0;
0072         for l=1:length(c.Data)
0073             for k=1:length(c.Data{l})
0074                 if strcmp(c.Data{l}, tables{k})
0075                     match = match + 1;
0076                 end
0077             end
0078         end
0079         close(c);
0080         if match == 5
0081             % we have an LTPDA database
0082             ltpda_dbs = [ltpda_dbs cellstr(db)];
0083         end
0084     end
0085
0086 end
0087 close(curs);
0088
0089 % If I opened the connection, close it again
0090 if wasEmpty
0091     close(conn);
0092 end
0093
0094 % Set the list
0095 id = findobj('Tag', 'dbSelect');
0096 set(id, 'String', ltpda_dbs, 'Value', 1);
0097
0098 % Get selected DB
0099 dbh = findobj('Tag', 'dbSelect');
0100 dbs = get(dbh, 'String');
0101 db = dbs{1};
0102
0103 % set selected DB
0104 dbh = findobj('Tag', 'databaseTxt');
0105 set(dbh, 'String', db);
0106
0107
0108

```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_importBtn.m](#)

cb_importBtn

PURPOSE ^

Callback executed when the user clicks on the import button.

SYNOPSIS ^

```
function cb_importBtn(varargin)
```

DESCRIPTION ^

```
Callback executed when the user clicks on the import button.
M Hewitson
$Id: cb_importBtn.m,v 1.3 2008/08/11 15:00:06 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

 [function \[ids, cids\] = getIds\(\)](#)

SOURCE CODE ^

```
0001 function cb_importBtn(varargin)
0002
0003 % Callback executed when the user clicks on the import button.
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_importBtn.m,v 1.3 2008/08/11 15:00:06 hewitson Exp $
0008 %
0009
0010 % Get connection
0011 mainfig = findobj('Tag', 'LTPDARepomainfig');
0012 conn = getappdata(mainfig, 'connection');
0013 if isempty(conn) || ~isa(conn, 'database')
0014     errorDlg('Please connect to a database first', 'No connection found');
0015     return
0016 end
0017
0018 % Get variable name specs
0019 h = findobj('Tag', 'objPrefixTxt');
0020 prefix = get(h, 'String');
0021 if isempty(prefix)
0022     prefix = 'obj';
0023     warning('! Using default prefix (obj)');
0024 end
0025 h = findobj('Tag', 'appendObjTypeChk');
0026 appendObj = get(h, 'Value');
0027
0028 % Get IDs from text box
0029 [ids, cids] = getIds();
0030
0031
0032 %-----
0033 % Retrieve these ids
0034 objs = [];
0035 for j=1:length(ids)
```

```

0036     disp(sprintf('+ retrieving object %d', ids(j)));
0037
0038     % determine object type
0039     tt = utils.mysql.getObjType(conn, ids(j));
0040     objname = sprintf('%s%03d', prefix, ids(j));
0041     if ~isempty(tt) && ~strcmp(tt, 'No Data')
0042         if appendObj
0043             objname = [objname '_' tt];
0044         end
0045     else
0046         error('!!! Object type is unknown. Does this object really exist?');
0047     end
0048
0049     % Retrieve object
0050     a = regexp(conn.URL, '///(\S+)/', 'tokens');
0051     db = regexp(conn.URL, '/', 'split');
0052     db = db{end};
0053     % add history
0054     pl = plist('hostname', a{1}, 'database', db, 'ID', ids(j), 'conn', conn);
0055     obj = eval(sprintf('%s(pl);', tt));
0056
0057     assignin('base', objname, obj);
0058     disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));
0059
0060     if j==1
0061         objs = {obj};
0062     else
0063         objs = [objs {obj}];
0064     end
0065 end
0066
0067 disp(sprintf('** Retrieved %d objects.', length(ids)));
0068
0069 %-----
0070 % Retrieve these Collections
0071 for k=1:length(cids)
0072
0073     % get Ids from Cid
0074     ids = utils.mysql.getObjIds(conn, cids(k));
0075     if isempty(ids)
0076         error('### This collection doesn't seem to exist.');

```

```
0122     ls = cs{j};
0123     [s,r] = strtok(ls);
0124     if ~isempty(s)
0125         if s(1) == 'c'
0126             s = s(2:end);
0127             cids = [cids round(str2num(s))];
0128         else
0129             ids = [ids round(str2num(s))];
0130         end
0131     end
0132     while ~isempty(r)
0133         [s,r] = strtok(r);
0134         if ~isempty(s)
0135             if s(1) == 'c'
0136                 s = s(2:end);
0137                 cids = [cids round(str2num(s))];
0138             else
0139                 ids = [ids round(str2num(s))];
0140             end
0141         end
0142     end
0143 end
0144
0145 end
```


cb_mainfigClose

PURPOSE ^

Callback executed when the main figure closes

SYNOPSIS ^

function cb_mainfigClose(varargin)

DESCRIPTION ^

Callback executed when the main figure closes
M Hewitson
\$Id: cb_mainfigClose.m,v 1.3 2008/02/29 10:53:14 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_mainfigClose(varargin)
0002
0003 % Callback executed when the main figure closes
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_mainfigClose.m,v 1.3 2008/02/29 10:53:14 hewitson Exp $
0008 %
0009
0010 % Close open connection
0011 conn = getappdata(varargin{1}, 'connection');
0012 if isa(conn, 'database')
0013     disp('* closing database connection');
0014     close(conn);
0015 end
0016
0017 disp('* Goodbye from the LTPDA Repository GUI *')
0018 delete(varargin{1})
0019
0020
0021
0022
```


cb_refreshObjsList

PURPOSE ^

Callback to refresh the list of objects.

SYNOPSIS ^

```
function cb_refreshObjsList(varargin)
```

DESCRIPTION ^

```
Callback to refresh the list of objects.  
M Hewitson  
$Id: cb_refreshObjsList.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [getWorkspaceObjs](#) Get a list of LTPDA objects in the MATLAB workspace
-  [setWorkspaceObjsList](#) Fill the workspace object list

This function is called by:

SOURCE CODE ^

```
0001 function cb_refreshObjsList(varargin)  
0002  
0003 % Callback to refresh the list of objects.  
0004 %  
0005 % M Hewitson  
0006 %  
0007 % $Id: cb_refreshObjsList.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $  
0008 %  
0009 %  
0010  
0011 % get a list of objects  
0012 objs = getWorkspaceObjs();  
0013  
0014 % set list  
0015 setWorkspaceObjsList(objs);  
0016  
0017
```


cb_refreshObjsListQL

PURPOSE ^

Callback to refresh the list of objects.

SYNOPSIS ^

function cb_refreshObjsListQL(varargin)

DESCRIPTION ^

Callback to refresh the list of objects.
M Hewitson
\$Id: cb_refreshObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [getWorkspaceObjs](#) Get a list of LTPDA objects in the MATLAB workspace
-  [setWorkspaceObjsListQL](#) Fill the workspace object list

This function is called by:

SOURCE CODE ^

```
0001 function cb_refreshObjsListQL(varargin)
0002
0003 % Callback to refresh the list of objects.
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_refreshObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
0008 %
0009 %
0010 % get a list of objects
0011 objs = getWorkspaceObjs();
0012
0013 % set list
0014 setWorkspaceObjsListQL(objs);
0015
0016
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_retrieveBtn.m](#)

cb_retrieveBtn

PURPOSE ^

Callback executed when the user clicks on the retrieve button.

SYNOPSIS ^

```
function cb_retrieveBtn(varargin)
```

DESCRIPTION ^

```
Callback executed when the user clicks on the retrieve button.
M Hewitson
$Id: cb_retrieveBtn.m,v 1.6 2008/02/29 10:53:49 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

 [function \[ids, cids\] = getIds\(\)](#)

SOURCE CODE ^

```
0001 function cb_retrieveBtn(varargin)
0002
0003 % Callback executed when the user clicks on the retrieve button.
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_retrieveBtn.m,v 1.6 2008/02/29 10:53:49 hewitson Exp $
0008 %
0009
0010 % Get connection
0011 mainfig = findobj('Tag', 'LTPDARepomainfig');
0012 conn = getappdata(mainfig, 'connection');
0013 if isempty(conn) || ~isa(conn, 'database')
0014     errordlg('Please connect to a database first', 'No connection found');
0015     return
0016 end
0017
0018 % Get variable name specs
0019 h = findobj('Tag', 'objPrefixTxt');
0020 prefix = get(h, 'String');
0021 if isempty(prefix)
0022     prefix = 'obj';
0023     warning('! Using default prefix (obj)');
0024 end
0025 h = findobj('Tag', 'appendObjTypeChk');
0026 appendObj = get(h, 'Value');
0027
0028 % Get IDs from text box
0029 [ids, cids] = getIds();
0030
0031
0032 %-----
0033 % Retrieve these ids
0034 objs = [];
0035 for j=1:length(ids)
```

```

0036     disp(sprintf('+ retrieving object %d', ids(j)));
0037
0038     % determine object type
0039     tt = mysql_getObjType(conn, ids(j));
0040     objname = sprintf('%s%03d', prefix, ids(j));
0041     if ~isempty(tt)
0042         if appendObj
0043             objname = [objname '_' tt];
0044         end
0045     else
0046         warning('!!! Object type is unknown');
0047     end
0048
0049     % Retrieve object
0050     a = regexp(conn.URL, '///(\S+)/', 'tokens');
0051     db = regexp(conn.URL, '/', 'split');
0052     db = db{end};
0053     % add history
0054     pl = plist('hostname', a{1}, 'database', db, 'ID', ids(j), 'conn', conn);
0055     obj = eval(sprintf('%s(pl);', tt));
0056
0057     assignin('base', objname, obj);
0058     disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));
0059
0060     if j==1
0061         objs = {obj};
0062     else
0063         objs = [objs {obj}];
0064     end
0065 end
0066
0067 disp(sprintf('** Retrieved %d objects.', length(ids)));
0068
0069 %-----
0070 % Retrieve these Collections
0071 for k=1:length(cids)
0072
0073     % get Ids from Cid
0074     ids = mysql_getObjIds(conn, cids(k));
0075
0076     for j=1:length(ids)
0077         disp(sprintf('+ retrieving collection %d : %d', cids(k), ids(j)));
0078         tt = mysql_getObjType(conn, ids(j));
0079         objname = sprintf('%sC%03d_%03d', prefix, cids(k), ids(j));
0080         if appendObj
0081             objname = [objname '_' tt];
0082         end
0083         % Retrieve object
0084         ipbits = regexp(conn.URL, '([0-9]+)', 'match');
0085         ip = [ipbits{1} '.' ipbits{2} '.' ipbits{3} '.' ipbits{4}];
0086         db = regexp(conn.URL, '/', 'split');
0087         db = db{end};
0088         % add history
0089         pl = plist('hostname', ip, 'database', db, 'ID', ids(j), 'conn', conn);
0090         obj = eval(sprintf('%s(pl);', tt));
0091
0092         assignin('base', objname, obj);
0093         disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));
0094         if j==1
0095             objs = {obj};
0096         else
0097             objs = [objs {obj}];
0098         end
0099     end
0100 end
0101
0102 disp(sprintf('** Retrieved %d objects.', length(ids)));
0103
0104
0105 end
0106
0107
0108 function [ids, cids] = getIds()
0109
0110     ids = [];
0111     cids = [];
0112
0113     th = findobj('Tag', 'retrieveIDsTxt');
0114     idStr = get(th, 'String');
0115     cs = cellstr(idStr);
0116
0117     for j=1:length(cs)
0118         disp('-----')
0119         ls = cs{j};
0120         [s,r] = strtok(ls);
0121         if ~isempty(s)

```



```
0122     if s(1) == 'c'
0123         s = s(2:end);
0124         cids = [cids round(str2num(s))];
0125     else
0126         ids = [ids round(str2num(s))];
0127     end
0128 end
0129 while ~isempty(r)
0130     [s,r] = strtok(r);
0131     if ~isempty(s)
0132         if s(1) == 'c'
0133             s = s(2:end);
0134             cids = [cids round(str2num(s))];
0135         else
0136             ids = [ids round(str2num(s))];
0137         end
0138     end
0139 end
0140 end
0141
0142 end
```

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_saveBtn.m](#)

cb_saveBtn

PURPOSE [^]

Callback executed when the user clicks on the save button.

SYNOPSIS [^]

```
function cb_saveBtn(varargin)
```

DESCRIPTION [^]

```
Callback executed when the user clicks on the save button.
M Hewitson
$Id: cb_saveBtn.m,v 1.3 2008/03/27 10:32:07 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

 [function \[ids, cids\] = getIds\(\)](#)

SOURCE CODE [^]

```
0001 function cb_saveBtn(varargin)
0002
0003 % Callback executed when the user clicks on the save button.
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_saveBtn.m,v 1.3 2008/03/27 10:32:07 hewitson Exp $
0008 %
0009
0010 % Get connection
0011 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0012 conn = getappdata(mainfig, 'connection');
0013 if isempty(conn) || ~isa(conn, 'database')
0014     errorlg('Please connect to a database first', 'No connection found');
0015     return
0016 end
0017
0018 % Get variable name specs
0019 h = findobj('Tag', 'objPrefixTxt');
0020 prefix = get(h, 'String');
0021 if isempty(prefix)
0022     prefix = 'obj';
0023     warning('! Using default prefix (obj)');
0024 end
0025 h = findobj('Tag', 'appendObjTypeChk');
0026 appendObj = get(h, 'Value');
0027
0028 % Get IDs from text box
0029 [ids, cids] = getIds();
0030
0031
0032 %-----
0033 % Retrieve these ids
0034 objs = [];
0035 for j=1:length(ids)
```

```

0036 disp(sprintf('+ retrieving object %d', ids(j)));
0037
0038 % determine object type
0039 tt = mysql_getObjType(conn, ids(j));
0040 objname = sprintf('%s%03d', prefix, ids(j));
0041 if ~isempty(tt) && ~strcmp(tt, 'No Data')
0042     if appendObj
0043         objname = [objname '_' tt];
0044     end
0045 else
0046     error('!!! Object type is unknown. Does this object really exist?');
0047 end
0048 % add file extension
0049 objname = [objname '.xml'];
0050
0051 % Retrieve object
0052 a = regexp(conn.URL, '//(\S+)/', 'tokens');
0053 db = regexp(conn.URL, '/', 'split');
0054 db = db{end};
0055 % add history
0056 pl = plist('hostname', a{1}, 'database', db, 'ID', ids(j), 'conn', conn);
0057 obj = eval(sprintf('%s(pl);', tt));
0058
0059 % write to disk
0060 save(obj, objname)
0061
0062 % assignin('base', objname, obj);
0063 disp(sprintf('** Retrieve object %d to disk [%s]', ids(j), objname));
0064
0065 if j==1
0066     objs = {obj};
0067 else
0068     objs = [objs {obj}];
0069 end
0070 end
0071
0072 disp(sprintf('** Retrieved %d objects.', length(ids)));
0073
0074 %-----
0075 % Retrieve these Collections
0076 for k=1:length(cids)
0077
0078     % get Ids from Cid
0079     ids = mysql_getObjIds(conn, cids(k));
0080     if isempty(ids)
0081         error('### This collection doesn't seem to exist.');
```

```
0122 idStr = get(th, 'String');
0123 cs = cellstr(idStr);
0124
0125 for j=1:length(cs)
0126     disp('-----')
0127     ls = cs{j};
0128     [s,r] = strtok(ls);
0129     if ~isempty(s)
0130         if s(1) == 'c'
0131             s = s(2:end);
0132             cids = [cids round(str2num(s))];
0133         else
0134             ids = [ids round(str2num(s))];
0135         end
0136     end
0137     while ~isempty(r)
0138         [s,r] = strtok(r);
0139         if ~isempty(s)
0140             if s(1) == 'c'
0141                 s = s(2:end);
0142                 cids = [cids round(str2num(s))];
0143             else
0144                 ids = [ids round(str2num(s))];
0145             end
0146         end
0147     end
0148 end
0149
0150 end
```

cb_serverSelect

PURPOSE ^

Callback executed when the user selects a server.

SYNOPSIS ^

function cb_serverSelect(varargin)

DESCRIPTION ^

Callback executed when the user selects a server.
M Hewitson
\$Id: cb_serverSelect.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_serverSelect(varargin)
0002
0003 % Callback executed when the user selects a server.
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_serverSelect.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0008 %
0009
0010 myh = varargin{1};
0011
0012 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0013
0014 % Get server string
0015 servers = get(myh, 'String');
0016 sid      = get(myh, 'Value');
0017
0018 server = servers{sid};
0019
0020 % Set server text field
0021
0022 stxth = findobj('Tag', 'serverTxt');
0023 set(stxth, 'String', server);
0024
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_submitBtn.m](#)

cb_submitBtn

PURPOSE ^

Callback executed when the user clicks on the 'Submit' button

SYNOPSIS ^

```
function cb_submitBtn(varargin)
```

DESCRIPTION ^

```
Callback executed when the user clicks on the 'Submit' button
M Hewitson
$Id: cb_submitBtn.m,v 1.7 2008/08/11 15:00:06 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_submitBtn(varargin)
0002
0003 % Callback executed when the user clicks on the 'Submit' button
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_submitBtn.m,v 1.7 2008/08/11 15:00:06 hewitson Exp $
0008 %
0009
0010 myh      = varargin{1};
0011 mainfig = varargin{4};
0012
0013 % get object list
0014 objlist = findobj('Tag', 'workspaceObjsList');
0015 objstr  = get(objlist, 'String');
0016 vals    = get(objlist, 'Value');
0017
0018 % Build sinfo structure
0019 sinfo.conn      = getappdata(mainfig, 'connection');
0020 sinfo.experiment_title      = get(findobj('Tag', 'experimentTitle'), 'String');
0021 sinfo.experiment_description = get(findobj('Tag', 'experimentDescription'), 'String');
0022 sinfo.analysis_description  = get(findobj('Tag', 'analysisDescription'), 'String');
0023 sinfo.quantity              = get(findobj('Tag', 'quantity'), 'String');
0024 sinfo.keywords              = get(findobj('Tag', 'keywords'), 'String');
0025 sinfo.reference_ids          = get(findobj('Tag', 'referenceIDs'), 'String');
0026 sinfo.additional_comments    = get(findobj('Tag', 'additionalComments'), 'String');
0027 sinfo.additional_authors     = get(findobj('Tag', 'additionalAuthors'), 'String');
0028
0029 if ~isa(sinfo.conn, 'database')
0030     error('### Please connect to a database before trying to submit.');
```

```
0043     for l=1:numel(obj)
0044         objs = [objs {obj(l)}];
0045         k = k + 1;
0046     end
0047 end
0048
0049 % Submit objects
0050 try
0051     [ids, cid] = ltpda_uo.submit(objs, sinfo);
0052     if isempty(ids)
0053         warning('!!! Failed to submit object: %s', objname);
0054     else
0055         disp(sprintf('++ Submitted object(s) %s (ref id = %s, collection id = %d)',
0056 objname, mat2str(ids), cid));
0057     end
0058 catch
0059     warning('!!! Failed to submit object: %s', objname);
0060     rethrow(lasterror)
0061 end
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_tableSelect.m](#)

cb_tableSelect

PURPOSE [^]

Callback executed when a table selection is made

SYNOPSIS [^]

```
function cb_tableSelect(varargin)
```

DESCRIPTION [^]

Callback executed when a table selection is made

M Hewitson

\$Id: cb_tableSelect.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [buildquery](#) Build a query from the settings on the query panel
-  [getFields](#) Get a list of database fields

This function is called by:

SOURCE CODE [^]

```
0001 function cb_tableSelect(varargin)
0002
0003 % Callback executed when a table selection is made
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_tableSelect.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0008 %
0009
0010 myh          = varargin{1};
0011 mainfig      = findobj('Tag', 'LTPDARepomainfig');
0012 fieldsList   = findobj('Tag', 'fieldsList');
0013 Nc           = getappdata(mainfig, 'Nconditions');
0014
0015 % Get conn
0016 conn = getappdata(mainfig, 'connection');
0017
0018 % Set fields in list
0019 fields = getFields(conn);
0020 set(fieldsList, 'Value', 1);
0021 set(fieldsList, 'String', fields);
0022
0023 % Set fields in condition drop downs
0024 for j=1:Nc
0025     cdbh = findobj('Tag', sprintf('c%02dField', j));
0026     set(cdbh, 'Value', 1);
0027     set(cdbh, 'String', fields);
0028 end
0029
0030 % Build query
0031 buildquery();
0032
0033 setappdata(mainfig, 'tableSelected', get(myh, 'Value'));
0034
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [cb_workspaceObjsList.m](#)

cb_workspaceObjsList

PURPOSE ^

Callback to fill the list of workspace objects

SYNOPSIS ^

```
function cb_workspaceObjsList(varargin)
```

DESCRIPTION ^

```
Callback to fill the list of workspace objects
M Hewitson
$Id: cb_workspaceObjsList.m,v 1.3 2008/02/23 13:44:08 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function cb_workspaceObjsList(varargin)
0002
0003 % Callback to fill the list of workspace objects
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_workspaceObjsList.m,v 1.3 2008/02/23 13:44:08 hewitson Exp $
0008 %
0009
0010
0011 myh      = varargin{1};
0012 mainfig = varargin{4};
0013
0014 % get the selected line
0015 val = get(myh, 'Value');
0016 if length(val) == 1
0017     lines = get(myh, 'String');
0018     objname = deblank(strtok(lines{val}));
0019
0020     cmd = sprintf('obj = evalin(''base'', '%s'');', objname);
0021     eval(cmd);
0022
0023     obj
0024     objtxt = display(obj);
0025 else
0026     objtxt = '';
0027 end
0028
0029 st = '';
0030 for j=1:length(objtxt)
0031     st = [st sprintf('%s\n', objtxt{j})];
0032 end
0033
0034 % set objDisplay Txt
0035 set(myh, 'TooltipString', st);
0036 % ltpdaObjInfo = findobj('Tag', 'ltpdaObjInfo');
0037 % set(ltpdaObjInfo, 'String', objtxt)
0038
0039
0040
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > cb_workspaceObjsListQL.m

cb_workspaceObjsListQL

PURPOSE [^]

Callback to fill the list of workspace objects

SYNOPSIS [^]

function cb_workspaceObjsListQL(varargin)

DESCRIPTION [^]

```
Callback to fill the list of workspace objects
M Hewitson
$Id: cb_workspaceObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [fillClassMethodsList](#) Callback to fill the list of class methods

This function is called by:

SOURCE CODE [^]

```
0001 function cb_workspaceObjsListQL(varargin)
0002
0003 % Callback to fill the list of workspace objects
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: cb_workspaceObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
0008 %
0009
0010
0011 myh      = varargin{1};
0012 mainfig = varargin{4};
0013
0014 % get the selected line
0015 val = get(myh, 'Value');
0016 if length(val) == 1
0017     lines = get(myh, 'String');
0018     objname = deblank(strtok(lines{val}));
0019
0020     cmd = sprintf('obj = evalin(''base'', '%s');', objname);
0021     eval(cmd);
0022
0023     objtxt = display(obj);
0024 else
0025     objtxt = '';
0026 end
0027
0028 % set objDisplay Txt
0029 ltpdaObjInfo = findobj('Tag', 'ltpdaObjInfoQL');
0030 set(ltpdaObjInfo, 'String', objtxt)
0031
0032 fillClassMethodsList();
0033
```


fillClassMethodsList

PURPOSE ^

Callback to fill the list of class methods

SYNOPSIS ^

function fillClassMethodsList(h)

DESCRIPTION ^

```
Callback to fill the list of class methods
M Hewitson
$Id: fillClassMethodsList.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [cb_workspaceObjsListQL](#) Callback to fill the list of workspace objects

SUBFUNCTIONS ^

 [function type = getClassFromString\(str\)](#)

SOURCE CODE ^

```
0001 function fillClassMethodsList(h)
0002
0003 % Callback to fill the list of class methods
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: fillClassMethodsList.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
0008 %
0009
0010
0011 % Get the object types
0012 id = findobj('Tag', 'workspaceObjsListQL');
0013 ostr = get(id, 'String');
0014 oidx = get(id, 'Value');
0015
0016 objStrs = ostr(oidx);
0017
0018 if ~isempty(objStrs)
0019
0020     type = getClassFromString(objStrs{1});
0021     diff = 0;
0022     for j=2:length(objStrs)
0023         if ~strcmp(type, getClassFromString(objStrs{j}))
0024             diff = 1;
0025         end
0026     end
0027
0028     mths = {};
0029     if ~diff
0030         tmps = methods(type);
0031         % Remove exceptions
0032         for j=1:length(tmps)
```

```
0033         if ~strcmp(tmps{j}, 'Contents')
0034 %             &&...
0035 %             ~strcmp(tmps{j}, 'subsref')
0036             mths = [mths tmps(j)];
0037         end
0038     end
0039 end
0040
0041 % Fill list
0042 id = findobj('Tag', 'classMethodList');
0043 set(id, 'Value', 1);
0044 set(id, 'String', mths);
0045
0046 end
0047
0048 function type = getClassFromString(str)
0049
0050 [s,r] = strtok(str, '(');
0051 type = r(2:end-1);
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [getFields.m](#)

getFields

PURPOSE ^

Get a list of database fields

SYNOPSIS ^

```
function fields = getFields(conn)
```

DESCRIPTION ^

```
Get a list of database fields
M Hewitson
$Id: getFields.m,v 1.3 2008/02/22 09:25:18 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [buildConditions](#) Callback to build the query conditions panel.
-  [cb_tableSelect](#) Callback executed when a table selection is made

SOURCE CODE ^

```
0001 function fields = getFields(conn)
0002
0003 % Get a list of database fields
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: getFields.m,v 1.3 2008/02/22 09:25:18 hewitson Exp $
0008 %
0009 %
0010
0011 % get the selected table
0012 tblSel = findobj('Tag', 'tableSelect');
0013 tbls = get(tblSel, 'String');
0014 tbl = tbls{get(tblSel, 'Value')};
0015
0016 if ~isempty(conn)
0017     % get the databases
0018     try
0019         q = sprintf('describe %s', tbl);
0020         curs = exec(conn, q);
0021         curs = fetch(curs);
0022         fields = curs.Data(:,1);
0023         close(curs);
0024     catch
0025         disp(sprintf('### Server returned: %s', curs.Message));
0026         warning('!!! Can not read from database');
0027         fields = {'-'};
0028     end
0029 else
0030     fields = {'-'};
0031 end
0032
```


[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [getTables.m](#)

getTables

PURPOSE [^]

Get a list of database tables

SYNOPSIS [^]

```
function tables = getTables(conn)
```

DESCRIPTION [^]

```
Get a list of database tables
M Hewitson
$Id: getTables.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [buildConditions](#) Callback to build the query conditions panel.

SOURCE CODE [^]

```
0001 function tables = getTables(conn)
0002
0003 % Get a list of database tables
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: getTables.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0008 %
0009
0010
0011 mainfig = findobj('Tag', 'LTPDAREpomainfig');
0012
0013 % Get conn
0014 conn = getappdata(mainfig, 'connection');
0015
0016 if ~isempty(conn)
0017     % get the databases
0018     q = 'show tables';
0019     curs = exec(conn, q);
0020     curs = fetch(curs);
0021
0022     tables = curs.Data;
0023     close(curs);
0024 else
0025     tables = {'-'};
0026 end
0027
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdaRepoGUI](#) > [callbacks](#) > [getWorkspaceObjs.m](#)

getWorkspaceObjs

PURPOSE [^]

Get a list of LTPDA objects in the MATLAB workspace

SYNOPSIS [^]

```
function objs = getWorkspaceObjs()
```

DESCRIPTION [^]

```
Get a list of LTPDA objects in the MATLAB workspace
M Hewitson
$Id: getWorkspaceObjs.m,v 1.3 2008/08/11 15:00:06 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

-  [cb_refreshObjsList](#) Callback to refresh the list of objects.
-  [cb_refreshObjsListQL](#) Callback to refresh the list of objects.

SOURCE CODE [^]

```
0001 function objs = getWorkspaceObjs()
0002
0003 % Get a list of LTPDA objects in the MATLAB workspace
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: getWorkspaceObjs.m,v 1.3 2008/08/11 15:00:06 hewitson Exp $
0008 %
0009 %
0010
0011 % get base workspace variables
0012 ws_vars = evalin('base','whos');
0013
0014 objs = [];
0015 for j=1:length(ws_vars)
0016
0017     cmd = sprintf('obj = evalin(''base'', '%s');', ws_vars(j).name);
0018     eval(cmd)
0019     if isa(obj, 'ltpda_uo')
0020         objs = [objs ws_vars(j)];
0021     end
0022
0023 end
0024
0025
0026
0027
```


setWorkspaceObjsList

PURPOSE ^

Fill the workspace object list

SYNOPSIS ^

function setWorkspaceObjsList(objs)

DESCRIPTION ^

Fill the workspace object list
M Hewitson
\$Id: setWorkspaceObjsList.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [cb_refreshObjsList](#) Callback to refresh the list of objects.

SOURCE CODE ^

```
0001 function setWorkspaceObjsList(objs)
0002
0003 % Fill the workspace object list
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: setWorkspaceObjsList.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0008 %
0009
0010
0011 id = findobj('Tag', 'workspaceObjsList');
0012
0013 objlist = [];
0014 for j=1:length(objs)
0015     obj = objs(j);
0016     str = sprintf('%s\t\t(%s)', obj.name, obj.class);
0017     objlist = [objlist cellstr(str)];
0018 end
0019
0020 set(id, 'Value', 1);
0021 set(id, 'String', objlist);
0022
0023
```


setWorkspaceObjsListQL

PURPOSE ^

Fill the workspace object list

SYNOPSIS ^

function setWorkspaceObjsListQL(objs)

DESCRIPTION ^

Fill the workspace object list
M Hewitson
\$Id: setWorkspaceObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [cb_refreshObjsListQL](#) Callback to refresh the list of objects.

SOURCE CODE ^

```
0001 function setWorkspaceObjsListQL(objs)
0002
0003 % Fill the workspace object list
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: setWorkspaceObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
0008 %
0009
0010 id = findobj('Tag', 'workspaceObjsListQL');
0011
0012 objlist = [];
0013 for j=1:length(objs)
0014     obj = objs(j);
0015     str = sprintf('%s\t\t(%s)', obj.name, obj.class);
0016     objlist = [objlist cellstr(str)];
0017 end
0018
0019 set(id, 'Value', 1);
0020 set(id, 'String', objlist);
0021
0022
```


Contents

PURPOSE ^

%% path: m/gui/ltpdalauncher
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/ltpdalauncher %%%%%%%%%%  
    <a href="matlab:help  
m/gui/ltpdalauncher/ltpda_launcher_close">m/gui/ltpdalauncher/ltpda_launcher_close</a>  
- % Callback executed when the main figure closes  
    <a href="matlab:help  
m/gui/ltpdalauncher/ltpda_launcher_launch_ltpdagui">m/gui/ltpdalauncher/ltpda_launcher_launch_ltpd  
- % Callback executed when the LTPDA GUI button is pressed  
    <a href="matlab:help  
m/gui/ltpdalauncher/ltpda_launcher_launch_pzmodel">m/gui/ltpdalauncher/ltpda_launcher_launch_pzm  
- % Callback executed when the LTPDA REPO GUI button is pressed  
    <a href="matlab:help  
m/gui/ltpdalauncher/ltpda_launcher_launch_repogui">m/gui/ltpdalauncher/ltpda_launcher_launch_rep  
- % Callback executed when the LTPDA REPO GUI button is pressed  
    <a href="matlab:help  
m/gui/ltpdalauncher/ltpdalauncher">m/gui/ltpdalauncher/ltpdalauncher</a> -  
presents the user with a control panel from which they can
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpda_launcher_close

PURPOSE ^

Callback executed when the main figure closes

SYNOPSIS ^

```
function ltpda_launcher_close(varargin)
```

DESCRIPTION ^

```
Callback executed when the main figure closes
M Hewitson
$Id: ltpda_launcher_close.m,v 1.1 2008/03/07 08:37:08 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_launcher_close(varargin)
0002
0003 % Callback executed when the main figure closes
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: ltpda_launcher_close.m,v 1.1 2008/03/07 08:37:08 hewitson Exp $
0008 %
0009
0010
0011 disp('* Goodbye from the LTPDA Launch Bay *')
0012 delete(varargin{1})
0013
0014
0015
0016 end
```


ltpda_launcher_launch_ltpdagui

PURPOSE ^

Callback executed when the LTPDA GUI button is pressed

SYNOPSIS ^

function ltpda_launcher_launch_ltpdagui(varargin)

DESCRIPTION ^

Callback executed when the LTPDA GUI button is pressed
M Hewitson
\$Id: ltpda_launcher_launch_ltpdagui.m,v 1.1 2008/03/07 08:37:08 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_launcher_launch_ltpdagui(varargin)
0002
0003     % Callback executed when the LTPDA GUI button is pressed
0004     %
0005     % M Hewitson
0006     %
0007     % $Id: ltpda_launcher_launch_ltpdagui.m,v 1.1 2008/03/07 08:37:08 hewitson Exp $
0008     %
0009
0010     ltpdagui;
0011
0012
0013 end
```


ltpda_launcher_launch_pzmodel

PURPOSE ^

Callback executed when the LTPDA REPO GUI button is pressed

SYNOPSIS ^

function ltpda_launcher_launch_pzmodel(varargin)

DESCRIPTION ^

Callback executed when the LTPDA REPO GUI button is pressed
M Hewitson
\$Id: ltpda_launcher_launch_pzmodel.m,v 1.1 2008/03/07 08:37:08 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_launcher_launch_pzmodel(varargin)
0002
0003     % Callback executed when the LTPDA REPO GUI button is pressed
0004     %
0005     % M Hewitson
0006     %
0007     % $Id: ltpda_launcher_launch_pzmodel.m,v 1.1 2008/03/07 08:37:08 hewitson Exp $
0008     %
0009
0010     pzmodel_helper;
0011
0012
0013 end
```


ltpda_launcher_launch_repogui

PURPOSE ^

Callback executed when the LTPDA REPO GUI button is pressed

SYNOPSIS ^

function ltpda_launcher_launch_repogui(varargin)

DESCRIPTION ^

Callback executed when the LTPDA REPO GUI button is pressed
M Hewitson
\$Id: ltpda_launcher_launch_repogui.m,v 1.1 2008/03/07 08:37:08 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_launcher_launch_repogui(varargin)
0002
0003     % Callback executed when the LTPDA REPO GUI button is pressed
0004     %
0005     % M Hewitson
0006     %
0007     % $Id: ltpda_launcher_launch_repogui.m,v 1.1 2008/03/07 08:37:08 hewitson Exp $
0008     %
0009
0010     repogui;
0011
0012
0013 end
```


ltpdalauncher

PURPOSE ^

LTPDALAUNCHER presents the user with a control panel from which they can

SYNOPSIS ^

function varargout = ltpdalauncher(varargin)

DESCRIPTION ^

```
LTPDALAUNCHER presents the user with a control panel from which they can
launch other LTPDA GUIs.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDALAUNCHER presents the user with a control panel from
              which they can launch other LTPDA GUIs.

CALL:        ltpdalauncher

VERSION:      $Id: ltpdalauncher.m,v 1.7 2008/05/11 11:13:26 hewitson Exp $
HISTORY:      07-03-08 M Hewitson
               Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

 [function call_explore_ao\(varargin\)](#)

SOURCE CODE ^

```
0001 function varargout = ltpdalauncher(varargin)
0002
0003 % LTPDALAUNCHER presents the user with a control panel from which they can
0004 % launch other LTPDA GUIs.
0005 %
0006 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0007 %
0008 % DESCRIPTION: LTPDALAUNCHER presents the user with a control panel from
0009 %               which they can launch other LTPDA GUIs.
0010 %
0011 % CALL:        ltpdalauncher
0012 %
0013 %
0014 %
0015 % VERSION:      $Id: ltpdalauncher.m,v 1.7 2008/05/11 11:13:26 hewitson Exp $
0016 %
0017 % HISTORY:      07-03-08 M Hewitson
0018 %               Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021
0022
```

```

0023 %% Check if I exist already
0024 hs = findall(0);
0025 found = -1;
0026 for j=1:length(hs)
0027     h = hs(j);
0028     if strcmp(get(h, 'Tag'), 'LTPDALauncher')
0029         found = h;
0030     end
0031 end
0032 if found ~= -1
0033     figure(found);
0034     return
0035 end
0036
0037 % id = findobj('Tag', 'LTPDALauncher');
0038 % if ~isempty(id)
0039 %     figure(id)
0040 %     return
0041 % end
0042
0043 %% Some initial setup
0044
0045 Gproperties.Gcol      = [240 240 240]/255;
0046 Gproperties.Gwidth    = 400;
0047 Gproperties.Gheight   = 400;
0048 Gproperties.Gborder   = 10;
0049 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0050
0051 Gproperties.Screen    = get(0, 'screensize');
0052 Gproperties.Gposition = [150 ...
0053     150 ...
0054     Gproperties.Gwidth...
0055     Gproperties.Gheight];
0056
0057 % Initialize and hide the GUI as it is being constructed.
0058 mainfig = figure('Name', 'LTPDA Launch Bay',...
0059     'NumberTitle', 'off',...
0060     'Visible', 'off',...
0061     'Position', Gproperties.Gposition,...
0062     'Color', Gproperties.Gcol,...
0063     'ToolBar', 'none',...
0064     'Resize', 'off',...
0065     'ToolBar', 'none',...
0066     'NextPlot', 'new',...
0067     'HandleVisibility', 'callback', ...
0068     'MenuBar', 'none',...
0069     'Tag', 'LTPDALauncher');
0070
0071 % Set mainfig callbacks
0072 set(mainfig, 'CloseRequestFcn', {'ltpda_launcher_close', mainfig});
0073
0074 % Set Application data
0075 setappdata(mainfig, 'Gproperties', Gproperties);
0076
0077 %% Add Logo
0078 M = 0.01;
0079 ax = axes('Position', [M M 1-2*M 1-2*M], 'Parent', mainfig);
0080 logo = imread('images/ltpda_blue.jpg');
0081 image(logo, 'Parent', ax);
0082 set(ax, 'Visible', 'on', 'Box', 'off', 'Xtick', [], 'Ytick', []);
0083 axis(ax, 'off');
0084
0085 %% Add launch buttons
0086
0087 BtnHeight = 20;
0088 BtnWidth = Gproperties.Gwidth - 20;
0089 BtnLeft = 10;
0090 BtnBottom = Gproperties.Gheight - BtnHeight - 10;
0091
0092 %-- LTPDA GUI
0093 BtnBottom = BtnBottom - BtnHeight - 10;
0094 pbh = uicontrol(mainfig, 'Style', 'pushbutton', 'String', 'LTPDA GUI',...
0095     'Position', [BtnLeft BtnBottom BtnWidth BtnHeight ],...
0096     'Tag', 'ltpda_launcher_ltpdagui');
0097 set(pbh, 'Callback', {'ltpda_launcher_launch_ltpdagui'});
0098
0099 %-- LTPDA REPO GUI
0100 pos = get(pbh, 'Position');
0101 BtnBottom = pos(2) - BtnHeight - 10;
0102 pbh = uicontrol(mainfig, 'Style', 'pushbutton', 'String', 'LTPDA REPO GUI',...
0103     'Position', [BtnLeft BtnBottom BtnWidth BtnHeight ],...
0104     'Tag', 'ltpda_launcher_repogui');
0105 set(pbh, 'Callback', {'ltpda_launcher_launch_repogui'});
0106
0107 %-- PZmodel helper
0108 pos = get(pbh, 'Position');

```

```

0109 BtnBottom = pos(2) - BtnHeight - 10;
0110 pbh = uicontrol(mainfig,'Style','pushbutton','String','pzmodel helper',...
0111     'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
0112     'Tag', 'ltpda_launcher_pzmodel');
0113 set(pbh, 'Callback', {'ltpda_launcher_launch_pzmodel'});
0114
0115 %-- specwin helper
0116 pos = get(pbh, 'Position');
0117 BtnBottom = pos(2) - BtnHeight - 10;
0118 pbh = uicontrol(mainfig,'Style','pushbutton','String','specwin GUI',...
0119     'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
0120     'Tag', 'ltpda_launcher_specwin');
0121 set(pbh, 'Callback', {'specwin_viewer'});
0122
0123 %-- constructor helper
0124 pos = get(pbh, 'Position');
0125 BtnBottom = pos(2) - BtnHeight - 10;
0126 pbh = uicontrol(mainfig,'Style','pushbutton','String','Constructor Helper',...
0127     'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
0128     'Tag', 'ltpda_constructor_helper');
0129 set(pbh, 'Callback', {'ltpda_constructor_helper'});
0130
0131 %-- object explorer
0132 pos = get(pbh, 'Position');
0133 BtnBottom = pos(2) - BtnHeight - 10;
0134 pbh = uicontrol(mainfig,'Style','pushbutton','String','Object Explorer',...
0135     'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
0136     'Tag', 'ltpda_object_explorer');
0137 set(pbh, 'Callback', {@call_explore_ao});
0138
0139 %-- quicklook
0140 pos = get(pbh, 'Position');
0141 BtnBottom = pos(2) - BtnHeight - 10;
0142 pbh = uicontrol(mainfig,'Style','pushbutton','String','Quicklook',...
0143     'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
0144     'Tag', 'ltpdaquicklook');
0145 set(pbh, 'Callback', {'ltpdaquicklook'});
0146
0147 %-- ltpdv
0148 pos = get(pbh, 'Position');
0149 BtnBottom = pos(2) - BtnHeight - 10;
0150 pbh = uicontrol(mainfig,'Style','pushbutton','String','ltpdv',...
0151     'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
0152     'Tag', 'ltpdv');
0153 set(pbh, 'Callback', {'ltpdv'});
0154
0155 %-- version string
0156 v = ver('LTPDA');
0157
0158 sth = uicontrol(mainfig,'Style','text',...
0159     'String',sprintf('%s %s', v.Name, v.Version),...
0160     'HorizontalAlignment','left',...
0161     'BackgroundColor',[130 190 250]/255, ...
0162     'ForegroundColor','w',...
0163     'Position',[10 10 120 25]);
0164
0165
0166
0167
0168 %% Start the GUI
0169
0170 % Make the GUI visible.
0171 set(mainfig,'Visible','on')
0172
0173
0174 function call_explore_ao(varargin)
0175     ltpda_explorer;
0176
0177 end
0178
0179
0180 end

```

Contents

PURPOSE ^

%% path: m/gui/ltpdv %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/ltpdv %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help m/gui/ltpdv/ltpdv">m/gui/ltpdv/ltpdv</a>                - % LTPDV
    dataviewer based on AOs and functions in LTPDA.
    <a href="matlab:help m/gui/ltpdv/make_ltpda_objects">m/gui/ltpdv/make_ltpda_objects</a> - %
    FSDATA
    <a href="matlab:help m/gui/ltpdv/scratch">m/gui/ltpdv/scratch</a>           - M-file for
    scratch.fig
    <a href="matlab:help m/gui/ltpdv/uitabpanel">m/gui/ltpdv/uitabpanel</a>       - adds a
    group of tabbed uipanel container objects to a figure.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpdv

PURPOSE ^

LTPDV dataviewer based on AOs and functions in LTPDA.

SYNOPSIS ^

function ltpdv(varargin)

DESCRIPTION ^

```
LTPDV dataviewer based on AOs and functions in LTPDA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDV dataviewer based on classes and methods in LTPDA.

VERSION:      $Id: ltpdv.m,v 1.7 2008/09/02 07:51:48 hewitson Exp $

HISTORY: 22-04-08 M Hewitson
          Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv(varargin)
0002 % LTPDV dataviewer based on AOs and functions in LTPDA.
0003 %
0004 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0005 %
0006 % DESCRIPTION: LTPDV dataviewer based on classes and methods in LTPDA.
0007 %
0008 %
0009 % VERSION:      $Id: ltpdv.m,v 1.7 2008/09/02 07:51:48 hewitson Exp $
0010 %
0011 %
0012 % HISTORY: 22-04-08 M Hewitson
0013 %          Creation
0014 %
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016 %
0017 %% Check if I exist already
0018 id = findobj('Tag', 'LTPDVmainfig');
0019 if ~isempty(id)
0020     figure(id)
0021     return
0022 end
0023
0024 %% Some initial setup
0025
0026 Gproperties.Gcol      = [240 240 240]/255;
0027 Gproperties.Gwidth    = 1100;
0028 Gproperties.Gheight   = 700;
0029 Gproperties.Gborder   = 10;
0030 Gproperties.fontsize  = 12;
0031
0032 Gproperties.Screen    = get(0, 'screensize');
```

```

0034 Gproperties.Gposition = [Gproperties.Screen(3)/2-Gproperties.Gwidth/2 ...
0035     Gproperties.Screen(4)/2-Gproperties.Gheight/2 ...
0036     Gproperties.Gwidth...
0037     Gproperties.Gheight];
0038
0039 % Initialize and hide the GUI as it is being constructed.
0040 mainfig = figure('Name', 'LTPDA Dataviewer',...
0041     'NumberTitle', 'off',...
0042     'Visible', 'off',...
0043     'Position', Gproperties.Gposition,...
0044     'Color', Gproperties.Gcol,...
0045     'ToolBar', 'none',...
0046     'MenuBar', 'none',...
0047     'Resize', 'on',...
0048     'Tag', 'LTPDVmainfig');
0049
0050 % Set mainfig callbacks
0051 set(mainfig, 'CloseRequestFcn', {'ltpdv_cb_mainfigClose', mainfig});
0052
0053 % Set Application data
0054 setappdata(mainfig, 'Gproperties', Gproperties);
0055
0056 %% Setup GUI elements
0057
0058 Hmargin = Gproperties.Gborder/Gproperties.Gwidth;
0059 Vmargin = Gproperties.Gborder/Gproperties.Gheight;
0060 BottomMargin = 0.2;
0061 SplitPoint = 0.35;
0062
0063 %----- Object list
0064
0065 % Panel
0066 pleft = Hmargin;
0067 phght = 1-2*Vmargin;
0068 pbot = (1 - Vmargin - phght);
0069 pwid = SplitPoint - 2*Hmargin;
0070 ph = uipanel(mainfig, 'Title', 'Object List', 'FontSize', Gproperties.fontsize,...
0071     'BackgroundColor', [0.6 0.7 0.9],...
0072     'Position', [pleft pbot pwid phght], 'Units', 'normalized');
0073
0074 % Object list
0075 pmarg = 0.025;
0076 Lleft = pmarg;
0077 Lbottom = 0.4+pmarg;
0078 Lwidth = 1-2*pmarg;
0079 Lheight = 1-Lbottom-pmarg;
0080 th = uitable(ph, ...
0081     'Units', 'normalized',...
0082     'ColumnName', {'Name', 'Type'}, ...
0083     'ColumnWidth', {220, 100}, ...
0084     'ColumnEditable', [true false], ...
0085     'Position', [Lleft Lbottom Lwidth Lheight],...
0086     'Tag', 'LTPDVobjList', ...
0087     'CellSelectionCallback', {@ltpdv_objectList_select, mainfig}, ...
0088     'CellEditCallback', {@ltpdv_objectList_edit, mainfig});
0089
0090 % Set column widths
0091 set(th, 'Units', 'pixel');
0092 pos = get(th, 'Position');
0093 set(th, 'Units', 'normalized');
0094 set(th, 'ColumnWidth', {0.7*pos(3), 0.2*pos(3)});
0095
0096 % Delete objects button
0097 pleft = pmarg;
0098 phght = 0.05;
0099 pbott = Lbottom - phght - pmarg;
0100 pwidt = 0.2;
0101 pbh = uicontrol(ph, 'Style', 'pushbutton',...
0102     'String', 'Delete',...
0103     'Callback', {@ltpdv_delete_objects, mainfig}, ...
0104     'Units', 'normalized', ...
0105     'Position', [pleft pbott pwidt phght]);
0106
0107 % Clear objects button
0108 pleft = pmarg + pwidt + pmarg;
0109 phght = 0.05;
0110 pbott = Lbottom - phght - pmarg;
0111 pwidt = 0.2;
0112 pbh = uicontrol(ph, 'Style', 'pushbutton',...
0113     'String', 'Clear',...
0114     'Callback', {@ltpdv_delete_objects, mainfig, 'all'}, ...
0115     'Units', 'normalized', ...
0116     'Position', [pleft pbott pwidt phght]);
0117
0118 % Save objects button
0119 pleft = pleft + pwidt + pmarg;

```

```

0120 phght = 0.05;
0121 pbott = Lbottom - phght - pmarg;
0122 pwid = 0.2;
0123 pbh = uicontrol(ph, 'Style', 'pushbutton', ...
0124 'String', 'Save', ...
0125 'Callback', {@ltpdv_save_objects, mainfig}, ...
0126 'Units', 'normalized', ...
0127 'Position', [pleft pbott pwid phght]);
0128
0129 % Load objects button
0130 pleft = pleft + pwid + pmarg;
0131 phght = 0.05;
0132 pbott = Lbottom - phght - pmarg;
0133 pwid = 0.2;
0134 pbh = uicontrol(ph, 'Style', 'pushbutton', ...
0135 'String', 'Load', ...
0136 'Callback', {@ltpdv_load_objects, mainfig}, ...
0137 'Units', 'normalized', ...
0138 'Position', [pleft pbott pwid phght]);
0139
0140 % Export objects button
0141 pleft = pmarg;
0142 phght = 0.05;
0143 pbott = pbott - phght - pmarg;
0144 pwid = 0.2;
0145 pbh = uicontrol(ph, 'Style', 'pushbutton', ...
0146 'String', 'Export', ...
0147 'Callback', {@ltpdv_export_objects, mainfig}, ...
0148 'Units', 'normalized', ...
0149 'Position', [pleft pbott pwid phght]);
0150
0151 % Explore objects button
0152 pleft = pleft + pwid + pmarg;
0153 pbh = uicontrol(ph, 'Style', 'pushbutton', ...
0154 'String', 'Explore', ...
0155 'Callback', {@ltpdv_explore_objects, mainfig}, ...
0156 'Units', 'normalized', ...
0157 'Position', [pleft pbott pwid phght]);
0158
0159
0160 %-----
0161 % Plot button
0162 bw = 0.2;
0163 bl = pmarg;
0164 bh = 0.05;
0165 bb = pmarg;
0166
0167 pbh = uicontrol(ph, 'Style', 'pushbutton', ...
0168 'String', 'Plot', ...
0169 'Callback', {@ltpdv_plot, mainfig}, ...
0170 'Units', 'normalized', ...
0171 'Position', [bl bb bw bh]);
0172
0173 % Plot history button
0174 bw = 0.2;
0175 bl = bl+bw+pmarg;
0176 bh = 0.05;
0177 bb = pmarg;
0178
0179 pbh = uicontrol(ph, 'Style', 'pushbutton', ...
0180 'String', 'Plot History', ...
0181 'Callback', {@ltpdv_hist_plot, mainfig}, ...
0182 'Units', 'normalized', ...
0183 'Position', [bl bb bw bh]);
0184
0185
0186 %----- TAB PANEL
0187 PanLeft = SplitPoint;
0188 Tleft = PanLeft + Hmargin;
0189 Theight = 1-2*Vmargin;
0190 Tbottom = (1 - Vmargin - Theight);
0191 Twidth = (1-PanLeft)-2*Hmargin;
0192
0193 htab = uitabgroup('Parent', mainfig, ...
0194 'Units', 'normalized', ...
0195 'Position', [Tleft Tbottom Twidth Theight], 'SelectionChangeFcn',
0196 {@ltpdv_tabChanged});
0197 t1 = uitab(htab, 'title', 'Import');
0198 t2 = uitab(htab, 'title', 'Math');
0199 t3 = uitab(htab, 'title', 'Preprocessing');
0200 t4 = uitab(htab, 'title', 'Filter');
0201 t5 = uitab(htab, 'title', 'Spectral');
0202 t6 = uitab(htab, 'title', 'Object Properties');
0203 t7 = uitab(htab, 'title', 'Plot');
0204
0205 % Set app data

```

```
0205 setappdata(mainfig, 'LTPDAobjects', {});
0206 setappdata(mainfig, 'ImportWorkspaceSelectedRows', []);
0207 setappdata(mainfig, 'ObjectListSelectedRows', []);
0208 setappdata(mainfig, 'Filters', []);
0209 setappdata(mainfig, 'FilterPoles', []);
0210 setappdata(mainfig, 'FilterZeros', []);
0211 setappdata(mainfig, 'TimesList', {});
0212
0213
0214 % Build panels
0215 ltpdv_buildImportPanel(t1, mainfig);
0216 ltpdv_buildMathPanel(t2, mainfig);
0217 ltpdv_buildPreprocessPanel(t3, mainfig);
0218 ltpdv_buildFilterPanel(t4, mainfig);
0219 ltpdv_buildSpectralPanel(t5, mainfig);
0220 ltpdv_buildPropertiesPanel(t6, mainfig);
0221 ltpdv_buildPlotPanel(t7, mainfig);
0222
0223
0224 %% Start the GUI
0225
0226 % Make the GUI visible.
0227 set(mainfig, 'Visible', 'on')
0228
0229
0230 end
0231
0232
0233
0234
0235
```

make_ltpda_objects

PURPOSE ^

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 clear all;
0002
0003 %% FSDATA
0004 p1 = pz(1);
0005 p2 = pz(4, 20);
0006 z1 = pz(20);
0007 pzm1 = pzmodel(2.3, [p1 p2], z1);
0008 r = resp(pzm1, plist('f1', 0.01, 'f2', 100, 'nf', 1000, 'scale', 'log'));
0009
0010 %% TSDATA
0011 fs = 10;
0012 nsecs = 1000;
0013
0014 ats = ao(plist('tsfcn', 'randn(size(t))', 'fs', fs, 'nsecs', nsecs));
0015
0016 %% CDATA
0017
0018 ac = ao(1:100);
0019
0020 %% SPECWIN
0021 s = specwin('Hanning', 1000);
0022
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [scratch.m](#)

scratch

PURPOSE [^]

SCRATCH M-file for scratch.fig

SYNOPSIS [^]

function varargout = scratch(varargin)

DESCRIPTION [^]

```
SCRATCH M-file for scratch.fig
SCRATCH, by itself, creates a new SCRATCH or raises the existing
singleton*.

H = SCRATCH returns the handle to a new SCRATCH or the handle to
the existing singleton*.

SCRATCH('CALLBACK',hObject,eventData,handles,...) calls the local
function named CALLBACK in SCRATCH.M with the given input arguments.

SCRATCH('Property','Value',...) creates a new SCRATCH or raises the
existing singleton*. Starting from the left, property value pairs are
applied to the GUI before scratch_OpeningFcn gets called. An
unrecognized property name or invalid value makes property application
stop. All inputs are passed to scratch_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

-  [function scratch_OpeningFcn\(hObject, eventdata, handles, varargin\)](#)
-  [function varargout = scratch_OutputFcn\(hObject, eventdata, handles\)](#)
-  [function popupmenu1_Callback\(hObject, eventdata, handles\)](#)
-  [function popupmenu1_CreateFcn\(hObject, eventdata, handles\)](#)
-  [function popupmenu2_Callback\(hObject, eventdata, handles\)](#)
-  [function popupmenu2_CreateFcn\(hObject, eventdata, handles\)](#)

SOURCE CODE [^]

```
0001 function varargout = scratch(varargin)
0002 % SCRATCH M-file for scratch.fig
0003 %     SCRATCH, by itself, creates a new SCRATCH or raises the existing
0004 %     singleton*.
0005 %
0006 %     H = SCRATCH returns the handle to a new SCRATCH or the handle to
0007 %     the existing singleton*.
0008 %
0009 %     SCRATCH('CALLBACK',hObject,eventData,handles,...) calls the local
0010 %     function named CALLBACK in SCRATCH.M with the given input arguments.
0011 %
0012 %     SCRATCH('Property','Value',...) creates a new SCRATCH or raises the
```



```

0013 %     existing singleton*. Starting from the left, property value pairs are
0014 %     applied to the GUI before scratch_OpeningFcn gets called. An
0015 %     unrecognized property name or invalid value makes property application
0016 %     stop. All inputs are passed to scratch_OpeningFcn via varargin.
0017 %
0018 %     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
0019 %     instance to run (singleton)".
0020 %
0021 % See also: GUIDE, GUIDATA, GUIHANDLES
0022 %
0023 % Edit the above text to modify the response to help scratch
0024 %
0025 % Last Modified by GUIDE v2.5 26-Aug-2008 16:10:20
0026 %
0027 % Begin initialization code - DO NOT EDIT
0028 gui_Singleton = 1;
0029 gui_State = struct('gui_Name',       mfilename, ...
0030                   'gui_Singleton',   gui_Singleton, ...
0031                   'gui_OpeningFcn',   @scratch_OpeningFcn, ...
0032                   'gui_OutputFcn',    @scratch_OutputFcn, ...
0033                   'gui_LayoutFcn',    [], ...
0034                   'gui_Callback',     []); ...
0035 if nargin && ischar(varargin{1})
0036     gui_State.gui_Callback = str2func(varargin{1});
0037 end
0038 %
0039 if nargout
0040     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
0041 else
0042     gui_mainfcn(gui_State, varargin{:});
0043 end
0044 % End initialization code - DO NOT EDIT
0045 %
0046 %
0047 % --- Executes just before scratch is made visible.
0048 function scratch_OpeningFcn(hObject, eventdata, handles, varargin)
0049 % This function has no output args, see OutputFcn.
0050 % hObject    handle to figure
0051 % eventdata  reserved - to be defined in a future version of MATLAB
0052 % handles    structure with handles and user data (see GUIDATA)
0053 % varargin   command line arguments to scratch (see VARARGIN)
0054 %
0055 % Choose default command line output for scratch
0056 handles.output = hObject;
0057 %
0058 % Update handles structure
0059 guidata(hObject, handles);
0060 %
0061 % UIWAIT makes scratch wait for user response (see UIRESUME)
0062 % uiwait(handles.figure1);
0063 %
0064 %
0065 % --- Outputs from this function are returned to the command line.
0066 function varargout = scratch_OutputFcn(hObject, eventdata, handles)
0067 % varargout  cell array for returning output args (see VARARGOUT);
0068 % hObject    handle to figure
0069 % eventdata  reserved - to be defined in a future version of MATLAB
0070 % handles    structure with handles and user data (see GUIDATA)
0071 %
0072 % Get default command line output from handles structure
0073 varargout{1} = handles.output;
0074 %
0075 %
0076 % --- Executes on selection change in popupmenu1.
0077 function popupmenu1_Callback(hObject, eventdata, handles)
0078 % hObject    handle to popupmenu1 (see GCBO)
0079 % eventdata  reserved - to be defined in a future version of MATLAB
0080 % handles    structure with handles and user data (see GUIDATA)
0081 %
0082 % Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
0083 %       contents{get(hObject,'Value')} returns selected item from popupmenu1
0084 %
0085 %
0086 % --- Executes during object creation, after setting all properties.
0087 function popupmenu1_CreateFcn(hObject, eventdata, handles)
0088 % hObject    handle to popupmenu1 (see GCBO)
0089 % eventdata  reserved - to be defined in a future version of MATLAB
0090 % handles    empty - handles not created until after all CreateFcns called
0091 %
0092 % Hint: popupmenu controls usually have a white background on Windows.
0093 %       See ISPC and COMPUTER.
0094 if ispc && isequal(get(hObject,'BackgroundColor'),
0095 get(0,'defaultUiControlBackgroundColor'))
0096     set(hObject,'BackgroundColor','white');
0097 end
0098 %

```

```
0098
0099 % --- Executes on selection change in popupmenu2.
0100 function popupmenu2_Callback(hObject, eventdata, handles)
0101 % hObject      handle to popupmenu2 (see GCBO)
0102 % eventdata    reserved - to be defined in a future version of MATLAB
0103 % handles      structure with handles and user data (see GUIDATA)
0104
0105 % Hints: contents = get(hObject,'String') returns popupmenu2 contents as cell array
0106 %          contents{get(hObject,'Value')} returns selected item from popupmenu2
0107
0108
0109 % --- Executes during object creation, after setting all properties.
0110 function popupmenu2_CreateFcn(hObject, eventdata, handles)
0111 % hObject      handle to popupmenu2 (see GCBO)
0112 % eventdata    reserved - to be defined in a future version of MATLAB
0113 % handles      empty - handles not created until after all CreateFcns called
0114
0115 % Hint: popupmenu controls usually have a white background on Windows.
0116 %       See ISPC and COMPUTER.
0117 if ispc && isequal(get(hObject,'BackgroundColor'),
0118 get(0,'defaultUiControlBackgroundColor'))
0118     set(hObject,'BackgroundColor','white');
0119 end
0120
0121
```

uitabpanel

PURPOSE ^

UITABPANEL adds a group of tabbed uipanel container objects to a figure.

SYNOPSIS ^

function hh = uitabpanel(varargin)

DESCRIPTION ^

UITABPANEL adds a group of tabbed uipanel container objects to a figure. The user can click on a tab with the left mouse button to select it. One tab is always selected, and appears on top of all the orthers. The syntax is the same as UIPANEL except the following property changes:

Property	Values
Title	a cell array of strings specifying the title of each tab
String	(same as Title)
TitlePosition	the location of tabs in relation of the visible panel, the value is the same as UIPANEL
Style	(same as TitlePosition)
FrameBackgroundColor	the background color for the tabpanel frame
FrameBorderType	the border type for the tabpanel frame
PanelBackgroundColor	the background color for the active tab/panel
TitleHighlightColor	the highlight color for the active tab
TitleForegroundColor	the foreground color for tab titiles
TitleBackgroundColor	the background color for the inactive tabs
SelectedItem	index specifying the currently active tab

A yet another type of UITABPANEL is a group of popup panels, which toggles open and close of a panel when its titile is clicked. Additional properties for popup tabpanel are:

Property	Values
Style	'popup'
PanelHeights	a numeric array specfying the height in characters of each popup panel
PanelBorderType	the border type for the popup panels
TitleHighlightColor	(not applicable)

By defining 'CreateFcn', MATLAB will build the group of panels at creation of UITABPANEL. An array of handles of the UIPANELs can be retrieved by calling getappdata(hTabpanel,'panels').

Example:
htab = uitabpanel('title',{'1','2','3'})
hpanel = getappdata(htab,'panels')

See also uitabdemo, uipanel, uicontrol, set, get.

Adapted for LTPDA by M Hewitson

\$Id: uitabpanel.m,v 1.2 2008/01/22 20:45:22 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

 function PopupTabCbk(hobj,evdt,indx)

-  `function PopupTabSliderCbk(hobj,evdt)`
-  `function PopupTabEnableAllCbk(hobj,evdt,cstr)`
-  `function PopupTabResizeCbk(hobj,evdt)`
-  `function TopBottomTabCbk(hobj,evdt,indx)`
-  `function TopBottomTabResizeCbk(hobj,evdt)`

SOURCE CODE [^]

```

0001 function hh = uitabpanel(varargin)
0002
0003
0004 % UITABPANEL adds a group of tabbed uipanel container objects to a figure.
0005 % The user can click on a tab with the left mouse button to select it.
0006 % One tab is always selected, and appears on top of all the others.
0007 % The syntax is the same as UIPANEL except the following property changes:
0008 %
0009 % Property Values
0010 % -----
0011 % Title a cell array of strings specifying the title of
0012 % each tab
0013 % String (same as Title)
0014 % TitlePosition the location of tabs in relation of the visible
0015 % panel, the value is the same as UIPANEL
0016 % Style (same as TitlePosition)
0017 % FrameBackgroundColor the background color for the tabpanel frame
0018 % FrameBorderType the border type for the tabpanel frame
0019 % PanelBackgroundColor the background color for the active tab/panel
0020 % TitleHighlightColor the highlight color for the active tab
0021 % TitleForegroundColor the foreground color for tab titles
0022 % TitleBackgroundColor the background color for the inactive tabs
0023 % SelectedItem index specifying the currently active tab
0024 %
0025 % A yet another type of UITABPANEL is a group of popup panels, which
0026 % toggles open and close of a panel when its title is clicked. Additional
0027 % properties for popup tabpanel are:
0028 %
0029 % Property Values
0030 % -----
0031 % Style 'popup'
0032 % PanelHeights a numeric array specifying the height in
0033 % characters of each popup panel
0034 % PanelBorderType the border type for the popup panels
0035 % TitleHighlightColor (not applicable)
0036 %
0037 % By defining 'CreateFcn', MATLAB will build the group of panels at
0038 % creation of UITABPANEL. An array of handles of the UIPANELS can
0039 % be retrieved by calling getappdata(hTabpanel,'panels').
0040 %
0041 % Example:
0042 % htab = uitabpanel('title',{'1','2','3'})
0043 % hpanel = getappdata(htab,'panels')
0044 %
0045 % See also uitabdemo, uipanel, uicontrol, set, get.
0046 %
0047 % Adapted for LTPDA by M Hewitson
0048 %
0049 % $Id: uitabpanel.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
0050 %
0051 %
0052 %
0053 %
0054 % Author: Shiyong Zhao (zhao@arch.umsi.edu)
0055 % Version: 1.0
0056 % First created: May 02, 2006
0057 % Last modified: May 20, 2006
0058 %
0059 tag = '';
0060 units = 'normalized';
0061 position = [0,0,1,1];
0062 fontweight = 'normal';
0063 [style,type] = deal('lefttop',2);
0064 visible = 'on';
0065
0066 foregroundcolor = [1,1,1];
0067 backgroundcolor = [0.7725,0.8431,0.9608];
0068 panelbackgroundcolor = [0.4314,0.5882,0.8431];
0069 titlehighlightcolor = [1,0.8,0];
0070
0071 horizontalalignment = 'center';
0072 [createfcn,deletefcn,resizefcn] = deal({},{});

```

```

0073
0074 active = 0;
0075 styles = {'popup','lefttop','righttop','leftbottom','rightbottom', ...
0076 'centertop','centerbottom'};
0077
0078 for k=1:2:nargin
0079     property = lower(varargin{k});
0080     value = varargin{k+1};
0081     switch property
0082     case {'style','tabposition','titleposition'}
0083         style = value;
0084         type = strmatch(lower(style),styles);
0085         if isempty(type)
0086             error(['Bad value for Style/TitlePosition property: ' style '.']);
0087         end
0088     case 'fontsize'
0089         fontsize = value;
0090     case {'title','string'}
0091         string = value;
0092         ntab = length(string);
0093     case {'frameposition','position'}
0094         position = value;
0095     case {'framebackgroundcolor','backgroundcolor'}
0096         backgroundcolor = value;
0097     case {'framebordertype','bordertype'}
0098         bordertype = value;
0099     case {'panelheights','heights'}
0100         panelheight = value;
0101     case {'titleforegroundcolor','foregroundcolor'}
0102         foregroundcolor = value;
0103     case {'selecteditem','active'}
0104         active = value;
0105     otherwise
0106         eval([property '= value;']);
0107     end
0108 end
0109
0110 if ~exist('parent','var')
0111     parent = gcf;
0112 end
0113 if ~exist('margins','var')
0114     if type==1
0115         margins = {[1.5,0.75,1.5,1],'characters'};
0116     else
0117         margins = {1,'pixels'};
0118     end
0119 end
0120 if ~iscell(margins)
0121     margins = {margins,units};
0122 end
0123
0124 if ~exist('bordertype','var')
0125     if type==1
0126         bordertype = 'etchedin';
0127     else
0128         bordertype = 'none';
0129     end
0130 end
0131
0132 if ~exist('titlebackgroundcolor','var')
0133     titlebackgroundcolor = 0.95*panelbackgroundcolor;
0134 end
0135
0136 cmenu = uicontextmenu;
0137 uimenu(cmenu,'Label','Goto Tab');
0138 uimenu(cmenu,'Separator','on');
0139
0140 htab = uipanel( ...
0141     'Parent',parent, ...
0142     'Units',units, ...
0143     'Position',position, ...
0144     'DeleteFcn',deletefcn, ...
0145     'ResizeFcn',resizefcn, ...
0146     'BackgroundColor',backgroundcolor, ...
0147     'BorderType',bordertype, ...
0148     'UIContextMenu',cmenu, ...
0149     'Visible',visible, ...
0150     'FontSize', fontsize,...
0151     'Tag',tag);
0152 set(htab,'Units','pixels');
0153 tabpos = get(htab,'Position');
0154
0155 status = uicontainer( ...
0156     'Parent',htab, ...
0157     'BackgroundColor',backgroundcolor, ...
0158     'Units',margins{2}, ...

```

```

0159 'Position',[1,1,1,1], ...
0160 'UIContextMenu',cmenu, ...
0161 'Visible','off');
0162 set(status,'Units','pixels');
0163 charsz = get(status,'Position');
0164 margins = [charsz(1:2),charsz(1:2)].*margins{1};
0165 set(status,'Units','characters');
0166 set(status,'Position',[1,1,1,1]);
0167 set(status,'Units','pixels');
0168 charsz = get(status,'Position');
0169 titleheight = 2*charsz(4);
0170
0171 if type==1
0172 %-----
0173 % PopupTab
0174 %-----
0175 panelpos = tabpos;
0176 titlepos = [margins(1),tabpos(2)+tabpos(4)-titleheight-margins(2), ...
0177 tabpos(3)-sum(margins([1,3])),titleheight];
0178 ttextpos = [0.05,0.1,0.9,0.76];
0179
0180 if ~exist('panelheight','var')
0181 panelheight = ones(1,ntab);
0182 end
0183 if ~exist('panelbordertype','var')
0184 panelbordertype = 'line';
0185 end
0186
0187 for k=1:ntab
0188 title(k) = uicontrol( ...
0189 'Parent',htab, ...
0190 'Units','pixels', ...
0191 'Position',titlepos, ...
0192 'ForegroundColor',foregroundcolor, ...
0193 'BackgroundColor',titlebackgroundcolor, ...
0194 'FontWeight',fontweight, ...
0195 'FontSize',fontsize, ...
0196 'HorizontalAlignment',horizontalalignment, ...
0197 'Callback',{@PopupTabCbK,k}, ...
0198 'Style','toggle', ...
0199 'String',string{k}, ...
0200 'Value',0, ...
0201 'Visible','on');
0202
0203 panel(k) = uipanel( ...
0204 'Parent',htab, ...
0205 'Units','pixels', ...
0206 'Position',titlepos+[0,0,0,panelheight(k)*charsz(4)], ...
0207 'BackgroundColor',panelbackgroundcolor, ...
0208 'BorderType',panelbordertype, ...
0209 'Visible','off');
0210
0211 uimenu(cmenu, ...
0212 'Label',string{k}, ...
0213 'Callback',{@PopupTabCbK,k});
0214 end
0215
0216 uimenu(cmenu,'Separator','on');
0217 uimenu(cmenu, ...
0218 'Label','Expand All', ...
0219 'Callback',{@PopupTabEnableAllCbK,'on'});
0220 uimenu(cmenu, ...
0221 'Label','Collapse All', ...
0222 'Callback',{@PopupTabEnableAllCbK,'off'});
0223
0224 sliderpos = [tabpos(3)-14,2,15,tabpos(4)-3];
0225 slider = uicontrol( ...
0226 'Parent',htab, ...
0227 'Units','pixels', ...
0228 'Position',sliderpos, ...
0229 'BackgroundColor',panelbackgroundcolor, ...
0230 'Style','slider', ...
0231 'Callback',@PopupTabSliderCbK, ...
0232 'Value',0, ...
0233 'Min',-1, ...
0234 'Max',0, ...
0235 'Visible','off');
0236
0237 statuspos = titlepos;
0238 statuspos([2,4]) = [margins(4),panelpos(4)-(ntab+1)*(margins(4)+titlepos(4))];
0239 set(status,'Position',statuspos);
0240 PopupTabCbK(title(1),[],active);
0241 set(status,'ResizeFcn',@PopupTabResizeCbK);
0242
0243 else
0244 %-----

```



```

0245 % TopT/Bottom Tab
0246 %-----
0247 margins(3:4) = -(margins(1:2)+margins(3:4));
0248 switch type
0249     case 2 %lefttop
0250         [loop,sign] = deal(1:ntab,[1,0,1,0]);
0251     case 3 %righttop
0252         [loop,sign] = deal(ntab:-1:1,[ntab,0,0,0]);
0253     case 4 %lefttbottom
0254         [loop,sign] = deal(1:ntab,[1,0,1,1]);
0255     case 5 %rightbottom
0256         [loop,sign] = deal(ntab:-1:1,[ntab,0,0,1]);
0257     case 6 %centertop
0258         [loop,sign] = deal(1:ntab,[1,ntab,1,0]);
0259     case 7 %centerbottom
0260         [loop,sign] = deal(1:ntab,[1,ntab,1,1]);
0261 end
0262 addtitle = 2*[sign(3:4),-1,-1];
0263 if sign(3)
0264     ttextpos = [0.05,0.08,0.9,0.76];
0265 else
0266     ttextpos = [0.05,0.05,0.9,0.76];
0267 end
0268
0269 if ~exist('panelbordertype','var')
0270     panelbordertype = 'beveledout';
0271 end
0272
0273 for k=1:ntab
0274     title(k) = uipanel( ...
0275         'Parent',htab, ...
0276         'Units','pixels', ...
0277         'BackgroundColor',titlebackgroundcolor, ...
0278         'BorderType',panelbordertype, ...
0279         'ButtonDownFcn',{@TopBottomTabCbk,k}, ...
0280         'Visible','on');
0281     %titlewidth(k) = (length(string{k})+4)*1.2*charsz(3);
0282
0283     ttext(k) = uicontrol( ...
0284         'Parent',title(k), ...
0285         'Units','normalized', ...
0286         'Position',ttextpos, ...
0287         'ForegroundColor',foregroundcolor, ...
0288         'BackgroundColor',titlebackgroundcolor, ...
0289         'HorizontalAlignment',horizontalalignment, ...
0290         'Style','text', ...
0291         'String',string{k}, ...
0292         'Enable','inactive', ...
0293         'FontSize',fontsize, ...
0294         'HitTest','off', ...
0295         'Visible','on');
0296
0297     panel(k) = uipanel( ...
0298         'Parent',htab, ...
0299         'Units','pixels', ...
0300         'BackgroundColor',panelbackgroundcolor, ...
0301         'BorderType',panelbordertype, ...
0302         'Visible','off');
0303
0304     uimenu(cmenu, ...
0305         'Label',string{k}, ...
0306         'Callback',{@TopBottomTabCbk,k});
0307 end
0308
0309 if ~active, active = 1; end
0310 set(panel(active),'Visible','on');
0311
0312 inset(1) = uicontrol( ...
0313     'Parent',htab, ...
0314     'Units','pixels', ...
0315     'BackgroundColor',panelbackgroundcolor, ...
0316     'Style','text', ...
0317     'Visible','on');
0318 if ~isnan(titlehighlightcolor)
0319     inset(2) = uicontrol( ...
0320         'Parent',htab, ...
0321         'Units','pixels', ...
0322         'BackgroundColor',titlehighlightcolor, ...
0323         'Style','text', ...
0324         'Visible','on');
0325     inset(3) = uicontrol( ...
0326         'Parent',htab, ...
0327         'Units','pixels', ...
0328         'BackgroundColor',0.85*titlehighlightcolor, ...
0329         'Style','text', ...
0330         'Visible','on');

```

```

0331 end
0332
0333 TopBottomTabResizeCbk(status,[]);
0334 set(status,'ResizeFcn',@TopBottomTabResizeCbk);
0335 end
0336
0337 setappdata(htab,'panels',panel);
0338 setappdata(htab,'status',status);
0339 if ~isempty(createfcn)
0340     createfcn(htab,[],panel,status);
0341 end
0342
0343 if narginout, hh = htab; end
0344
0345
0346 %-----
0347 % Inner Functions for Popup Tab
0348 %-----
0349 function PopupTabCbk(hobj,evdt,indx)
0350
0351     set(title,'Value',0);
0352     if indx>0
0353         indxvis = strcmp(get(panel(indx),'Visible'),'off');
0354         if indxvis
0355             set(panel(indx),'Visible','on');
0356         else
0357             set(panel(indx),'Visible','off');
0358         end
0359     end
0360     if isempty(strmatch('on',get(panel,'Visible')))
0361         set(status,'Visible','on');
0362     else
0363         set(status,'Visible','off');
0364     end
0365
0366     currval = get(slider,'Value');
0367     if currval<0
0368         set(slider,'Value',0);
0369         PopupTabSliderCbk(slider,evdt);
0370     end
0371     titlepos = get(title(1),'Position');
0372     addtolen = titlepos(2)+titlepos(4)+margins(4);
0373
0374     for k=1:ntab
0375         titlepos = get(title(k),'Position');
0376         titlepos(2) = addtolen-titlepos(4)-margins(4);
0377         set(title(k),'Position',titlepos);
0378         addtolen = titlepos(2);
0379         visible = strcmp(get(panel(k),'Visible'),'on');
0380         if visible
0381             panelpos = get(panel(k),'Position');
0382             addtolen = addtolen-panelpos(4);
0383             panelpos(2) = addtolen;
0384             set(panel(k),'Position',panelpos);
0385         end
0386     end
0387
0388     if visible
0389         botpos = get(panel(ntab),'Position');
0390     else
0391         botpos = get(title(ntab),'Position');
0392     end
0393
0394     visible = strcmp(get(slider,'Visible'),'on');
0395     addwidth = (-1)^(~visible);
0396     if botpos(2)*addwidth>0
0397         if visible
0398             set(slider,'Visible','off');
0399         else
0400             set(slider,'Visible','on');
0401         end
0402     else
0403         if visible
0404             addwidth = 0;
0405         else
0406             return;
0407         end
0408     end
0409
0410     addwidth = addwidth*sliderpos(3);
0411     for k=1:ntab
0412         titlepos = get(title(k),'Position');
0413         titlepos(3) = titlepos(3)+addwidth;
0414         set(title(k),'Position',titlepos);
0415         set(title(k),'UserData',titlepos(2));
0416     end

```



```

0417     panelpos = get(panel(k), 'Position');
0418     panelpos(3) = panelpos(3)+addwidth;
0419     set(panel(k), 'Position', panelpos);
0420     set(panel(k), 'UserData', panelpos(2));
0421 end
0422
0423     toplen = botpos(2)-margins(2);
0424     if toplen>0
0425         set(slider, 'Value', 0);
0426         return;
0427     end
0428
0429     try
0430         if indxvis
0431             viewpos = get(panel(indx), 'Position');
0432             if viewpos(2)-viewpos(4)<0
0433                 if viewpos(4)+margins(4)<=tabpos(4);
0434                     currval = min(0, viewpos(2)-margins(4)/2);
0435                 else
0436                     viewpos = get(title(indx), 'Position');
0437                     currval = viewpos(2)+viewpos(4)+margins(4)/2 - tabpos(4);
0438                 end
0439             end
0440         end
0441     catch
0442     end
0443
0444     set(slider, 'Min', toplen);
0445     set(slider, 'Value', max(currval, toplen));
0446     set(slider, 'SliderStep', [0.3, 1]/max(1, -toplen/100));
0447     PopupTabSliderCbK(slider, evdt);
0448
0449 end
0450
0451
0452 %-----
0453 function PopupTabSliderCbK(hobj, evdt)
0454
0455     currval = get(hobj, 'Value');
0456     for k=1:ntab
0457         titlepos = get(title(k), 'Position');
0458         titlepos(2) = get(title(k), 'UserData')-currval;
0459         set(title(k), 'Position', titlepos);
0460         panelpos = get(panel(k), 'Position');
0461         panelpos(2) = get(panel(k), 'UserData')-currval;
0462         set(panel(k), 'Position', panelpos);
0463     end
0464
0465 end
0466
0467 %-----
0468 function PopupTabEnableAllCbK(hobj, evdt, cstr)
0469
0470     set(panel, 'Visible', cstr);
0471     PopupTabCbK(hobj, evdt, 0);
0472
0473 end
0474
0475 %-----
0476 function PopupTabResizeCbK(hobj, evdt)
0477
0478     oldpos = tabpos;
0479     units = get(htab, 'Units');
0480     set(htab, 'Units', 'pixels');
0481     tabpos = get(htab, 'Position');
0482     set(htab, 'Units', units);
0483
0484     addtopos = tabpos-oldpos;
0485     sliderpos([1, 2, 4]) = sliderpos([1, 2, 4])+addtopos([1, 2, 4]);
0486     set(slider, 'Position', sliderpos);
0487
0488     addtopos = [tabpos(3:4)-oldpos(3:4), 0, 0];
0489     currval = get(slider, 'Value');
0490     if currval<0
0491         set(slider, 'Value', 0);
0492         PopupTabSliderCbK(slider, evdt);
0493     end
0494
0495     for k=1:ntab
0496         titlepos = get(title(k), 'Position');
0497         set(title(k), 'Position', titlepos+addtopos);
0498     end
0499     PopupTabCbK(title(1), evdt, 0);
0500
0501
0502

```

```

0503     visible = strcmp(get(slider, 'Visible'), 'on');
0504     if visible & currval<0
0505         set(slider, 'Value', max(currval, get(slider, 'Min')));
0506         PopupTabSliderCbk(slider, evdt);
0507     end
0508
0509 end
0510
0511 %-----
0512 % Inner Functions for Top/Bottom Tab
0513 %-----
0514
0515 function TopBottomTabCbk(hobj, evdt, indx)
0516
0517     if active==indx, return; end
0518
0519     % deactivate tab(active)
0520     n = active;
0521     titlepos = get(title(n), 'Position');
0522     titlepos = titlepos+addtitle.*[n==sign(1), 1, n==sign(1), 1];
0523     titlepos(3) = titlepos(3)+addtitle(3)*(n==sign(2));
0524     set(title(n), 'Position', titlepos, 'BackgroundColor', titlebackgroundcolor);
0525     set(ttext(n), 'FontWeight', 'normal', 'BackgroundColor', titlebackgroundcolor);
0526     set(panel(n), 'Visible', 'off');
0527
0528     % activate tab(indx)
0529     active = indx;
0530     titlepos = get(title(indx), 'Position');
0531     titlepos = titlepos-addtitle.*[indx==sign(1), 1, indx==sign(1), 1];
0532     titlepos(3) = titlepos(3)-addtitle(3)*(indx==sign(2));
0533
0534     set(title(indx), 'Position', titlepos, 'BackgroundColor', panelbackgroundcolor);
0535     set(ttext(indx), 'FontWeight', 'bold', 'BackgroundColor', panelbackgroundcolor);
0536     insetpos = [titlepos(1)+1, titlepos(2)+sign(4)*titlepos(4)-2, titlepos(3)-1, 3];
0537     set(inset(1), 'Position', insetpos);
0538     try
0539         insetpos = insetpos+[0, (1-2*sign(4))*titleheight+2*sign(4), 0, -2];
0540         set(inset(2), 'Position', insetpos);
0541         set(inset(3), 'Position', insetpos+[2, 1-2*sign(4), -4, 0]);
0542     catch
0543     end
0544     set(panel(indx), 'Visible', 'on');
0545
0546     if isappdata(htab, 'SelectionChangeFcn')
0547         SelectionChangeFcn = getappdata(htab, 'SelectionChangeFcn');
0548         eventdata = struct('EventName', 'SelectionChanged', '...',
0549             'OldValue', n, 'NewValue', indx, 'Panels', panel, 'Status', status);
0550         if iscell(SelectionChangeFcn)
0551             SelectionChangeFcn{1}(hobj, eventdata, SelectionChangeFcn{2:end});
0552         else
0553             SelectionChangeFcn(hobj, eventdata);
0554         end
0555     end
0556
0557 end
0558
0559 %-----
0560 function TopBottomTabResizeCbk(hobj, evdt)
0561
0562     units = get(htab, 'Units');
0563     set(htab, 'Units', 'pixels');
0564     tabpos = get(htab, 'Position');
0565     set(htab, 'Units', units);
0566
0567     titlepos = [0, 0, 0, titleheight];
0568     if sign(4)
0569         panelpos = [0, titleheight+1, tabpos(3), tabpos(4)-titleheight]+margins;
0570         titlepos(2) = panelpos(2)-titlepos(4);
0571     else
0572         panelpos = [0, 0, tabpos(3), tabpos(4)-titleheight]+margins;
0573         titlepos(2) = panelpos(2)+panelpos(4);
0574     end
0575     set(panel, 'Position', panelpos);
0576
0577     n = sum(cellfun('length', string))+2*ntab;
0578     if type<6
0579         if (n+2*ntab)*1.2*charsz(3)+2>panelpos(3)
0580             [loop, sign(1:3)] = deal(1:ntab, [1, ntab, 1]);
0581         else
0582             if type==2 | type==4
0583                 [loop, sign(1:3)] = deal(1:ntab, [1, 0, 1]);
0584             else
0585                 [loop, sign(1:3)] = deal(ntab:-1:1, [ntab, 0, 0]);
0586             end
0587         end
0588     end

```

```

0589     addtitle = 2*[sign(3:4), -1, -1];
0590 end
0591 titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
0592 for k=loop
0593     if sign(2)
0594         titlepos(3) = panelpos(3)*(length(string{k})+2)/n;
0595         addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
0596     else
0597         titlepos(3) = (length(string{k})+4)*1.2*charsz(3);
0598         addtopos = addtitle.*[k==sign(1),1,k==sign(1),1];
0599     end
0600     titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
0601     set(title(k), 'Position', titlepos+addtopos);
0602     titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
0603 end
0604
0605 statuspos = titlepos;
0606 statuspos(1) = sign(3)*titlepos(1)+2;
0607 statuspos(3) = titlepos(1)+sign(3)*(tabpos(3)-2*titlepos(1))-4;
0608
0609 titlepos = get(title(active), 'Position');
0610 titlepos = titlepos-addtitle.*[active==sign(1),1,active==sign(1),1];
0611 titlepos(3) = titlepos(3)-addtitle(3)*(active==sign(2));
0612 set(title(active), 'Position', titlepos, 'BackgroundColor', panelbackgroundcolor);
0613 set(ttext(active), 'FontWeight', 'bold', 'BackgroundColor', panelbackgroundcolor);
0614
0615 insetpos = [titlepos(1)+1, titlepos(2)+sign(4)*titlepos(4)-2, titlepos(3)-1, 3];
0616 set(inset(1), 'Position', insetpos);
0617 try
0618     insetpos = insetpos+[0, (1-2*sign(4))*titleheight+2*sign(4), 0, -2];
0619     set(inset(2), 'Position', insetpos);
0620     set(inset(3), 'Position', insetpos+[2, 1-2*sign(4), -4, 0]);
0621 catch
0622 end
0623 try
0624     % This causes the above statements being excuted twice for each resizing.
0625     set(status, 'Position', statuspos, 'Visible', 'on');
0626 catch
0627     set(status, 'Visible', 'off');
0628 end
0629
0630 end
0631
0632 end

```

Contents

PURPOSE ^

%% path: m/gui/ltpdv/callbacks
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/ltpdv/callbacks %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_buildFilterPanel">m/gui/ltpdv/callbacks/ltpdv_buildFilterPanel</a>  
- % Add new sub tabbed panel  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_buildImportPanel">m/gui/ltpdv/callbacks/ltpdv_buildImportPanel</a>  
- % Build a panel to contain this tab  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_buildMathPanel">m/gui/ltpdv/callbacks/ltpdv_buildMathPanel</a>  
- Sizes  
    m/gui/ltpdv/callbacks/ltpdv_buildPlotPanel - (No help available)  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_buildPreprocessPanel">m/gui/ltpdv/callbacks/ltpdv_buildPreprocessPanel</a>  
- List of functions  
    m/gui/ltpdv/callbacks/ltpdv_buildPropertiesPanel - (No help available)  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_buildSpectralPanel">m/gui/ltpdv/callbacks/ltpdv_buildSpectralPanel</a>  
- List of functions  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_build_buildobject">m/gui/ltpdv/callbacks/ltpdv_build_buildobject</a>  
- callback for the build object button on the build  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_build_class_select">m/gui/ltpdv/callbacks/ltpdv_build_class_select</a>  
- callback when the user selects the class  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_build_get_selected_class">m/gui/ltpdv/callbacks/ltpdv_build_get_selected_class</a>  
- returns the currently selected class on  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_build_get_selected_pset">m/gui/ltpdv/callbacks/ltpdv_build_get_selected_pset</a>  
- returns the currently selected set on  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_build_params_panel">m/gui/ltpdv/callbacks/ltpdv_build_params_panel</a>  
- % we give a pop-up menu of sets  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_build_set_select">m/gui/ltpdv/callbacks/ltpdv_build_set_select</a>  
- callback when the user selects a parameter set to  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_calc_btn">m/gui/ltpdv/callbacks/ltpdv_calc_btn</a>  
- called when one of the calculator buttons is pushed.  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_cb_mainfigClose">m/gui/ltpdv/callbacks/ltpdv_cb_mainfigClose</a>  
- Callback executed when the main figure closes  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_delete_objects">m/gui/ltpdv/callbacks/ltpdv_delete_objects</a>  
- delete the selected objects from the object list  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_explore_objects">m/gui/ltpdv/callbacks/ltpdv_explore_objects</a>  
- Handles  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_export_objects">m/gui/ltpdv/callbacks/ltpdv_export_objects</a>  
- Handles  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_fcn_run">m/gui/ltpdv/callbacks/ltpdv_fcn_run</a>  
- % LTPDV_FCN_RUN runs the command on the selected panel  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_filterAdd">m/gui/ltpdv/callbacks/ltpdv_filterAdd</a>  
- Handles  
    <a href="matlab:help  
m/gui/ltpdv/callbacks/ltpdv_filterApply">m/gui/ltpdv/callbacks/ltpdv_filterApply</a>
```

- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_filterBrowse](matlab:help m/gui/ltpdv/callbacks/ltpdv_filterBrowse)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_filterDelete](matlab:help m/gui/ltpdv/callbacks/ltpdv_filterDelete)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_filterImport](matlab:help m/gui/ltpdv/callbacks/ltpdv_filterImport)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_filter_control_chans](matlab:help m/gui/ltpdv/callbacks/ltpdv_filter_control_chans)
- DV_FILTERCONTROLCHANS filter out control channels
 - [m/gui/ltpdv/callbacks/ltpdv_filter_from_standard_panel](matlab:help m/gui/ltpdv/callbacks/ltpdv_filter_from_standard_panel)
- builds a filter from the description on
 - [m/gui/ltpdv/callbacks/ltpdv_filter_get_pzmodel](matlab:help m/gui/ltpdv/callbacks/ltpdv_filter_get_pzmodel)
- poles
 - [m/gui/ltpdv/callbacks/ltpdv_filter_getf](matlab:help m/gui/ltpdv/callbacks/ltpdv_filter_getf)
- f1
 - [m/gui/ltpdv/callbacks/ltpdv_filter_list_select](matlab:help m/gui/ltpdv/callbacks/ltpdv_filter_list_select)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_filtersClear](matlab:help m/gui/ltpdv/callbacks/ltpdv_filtersClear)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_getWorkspaceObjs](matlab:help m/gui/ltpdv/callbacks/ltpdv_getWorkspaceObjs)
- gets a list of LTPDA user objects currently in the
 - [m/gui/ltpdv/callbacks/ltpdv_get_params_from_panel](matlab:help m/gui/ltpdv/callbacks/ltpdv_get_params_from_panel)
- get a plist from the parameters on a
 - [m/gui/ltpdv/callbacks/ltpdv_get_selected_filter](matlab:help m/gui/ltpdv/callbacks/ltpdv_get_selected_filter)
- selected panel
 - [m/gui/ltpdv/callbacks/ltpdv_get_selected_objs](matlab:help m/gui/ltpdv/callbacks/ltpdv_get_selected_objs)
- gets the selected objects from the object list.
 - [m/gui/ltpdv/callbacks/ltpdv_hist_plot](matlab:help m/gui/ltpdv/callbacks/ltpdv_hist_plot)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_importObjs](matlab:help m/gui/ltpdv/callbacks/ltpdv_importObjs)
- import given objects to the object list
 - [m/gui/ltpdv/callbacks/ltpdv_load_objects](matlab:help m/gui/ltpdv/callbacks/ltpdv_load_objects)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_objectList_edit](matlab:help m/gui/ltpdv/callbacks/ltpdv_objectList_edit)
- activates when the user changes the content of a
 - [m/gui/ltpdv/callbacks/ltpdv_objectList_select](matlab:help m/gui/ltpdv/callbacks/ltpdv_objectList_select)
- activates when the user selects cells in the
 - [m/gui/ltpdv/callbacks/ltpdv_parse_control_chans](matlab:help m/gui/ltpdv/callbacks/ltpdv_parse_control_chans)
- DV_PARSECONTROLCHANS parse out a list of control channels from the input
 - [m/gui/ltpdv/callbacks/ltpdv_plot](matlab:help m/gui/ltpdv/callbacks/ltpdv_plot)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_plotFilterResp](matlab:help m/gui/ltpdv/callbacks/ltpdv_plotFilterResp)
- plots the response of the selected filter.
 - [m/gui/ltpdv/callbacks/ltpdv_poleAdd](matlab:help m/gui/ltpdv/callbacks/ltpdv_poleAdd)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_poleClear](matlab:help m/gui/ltpdv/callbacks/ltpdv_poleClear)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_poleDelete](matlab:help m/gui/ltpdv/callbacks/ltpdv_poleDelete)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_preproc_fcnlist](matlab:help m/gui/ltpdv/callbacks/ltpdv_preproc_fcnlist)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_preproc_param_act](matlab:help m/gui/ltpdv/callbacks/ltpdv_preproc_param_act)
- Handles
 - [m/gui/ltpdv/callbacks/ltpdv_preproc_sets_callback](matlab:help m/gui/ltpdv/callbacks/ltpdv_preproc_sets_callback)
- (No help available)
- callback for apply button on the object properties
 - [m/gui/ltpdv/callbacks/ltpdv_properties_apply](matlab:help m/gui/ltpdv/callbacks/ltpdv_properties_apply)
- % Object class
 - [m/gui/ltpdv/callbacks/ltpdv_properties_build_props](matlab:help m/gui/ltpdv/callbacks/ltpdv_properties_build_props)


```

m/gui/ltpdv/callbacks/ltpdv_pzmResp">m/gui/ltpdv/callbacks/ltpdv_pzmResp</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_pzmStore">m/gui/ltpdv/callbacks/ltpdv_pzmStore</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_pzmStoreResp">m/gui/ltpdv/callbacks/ltpdv_pzmStoreResp</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_refresh_object_list">m/gui/ltpdv/callbacks/ltpdv_refresh_object_list</a>
- sets the object list from the app data.
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_save_objects">m/gui/ltpdv/callbacks/ltpdv_save_objects</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_search_channels">m/gui/ltpdv/callbacks/ltpdv_search_channels</a>
- search the channel list and reduce it to those that
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_addTime">m/gui/ltpdv/callbacks/ltpdv_server_addTime</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_channellist">m/gui/ltpdv/callbacks/ltpdv_server_channellist</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_clearTimes">m/gui/ltpdv/callbacks/ltpdv_server_clearTimes</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_datatype_select">m/gui/ltpdv/callbacks/ltpdv_server_datatype_select</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_deleteTimes">m/gui/ltpdv/callbacks/ltpdv_server_deleteTimes</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_getChannels">m/gui/ltpdv/callbacks/ltpdv_server_getChannels</a>
- Set channel list
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_getLatest">m/gui/ltpdv/callbacks/ltpdv_server_getLatest</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_channels">m/gui/ltpdv/callbacks/ltpdv_server_get_channels</a>
- handle to list
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_comment">m/gui/ltpdv/callbacks/ltpdv_server_get_comment</a>
- handle
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_data">m/gui/ltpdv/callbacks/ltpdv_server_get_data</a>
- times
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_latest_chans">m/gui/ltpdv/callbacks/ltpdv_server_get_latest_chans</a>
- server details
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_server">m/gui/ltpdv/callbacks/ltpdv_server_get_server</a>
- data type
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_start_time">m/gui/ltpdv/callbacks/ltpdv_server_get_start_time</a>
- Handle
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_stop_time">m/gui/ltpdv/callbacks/ltpdv_server_get_stop_time</a>
- Handle
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_times">m/gui/ltpdv/callbacks/ltpdv_server_get_times</a>
- Do we have a list?
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_import">m/gui/ltpdv/callbacks/ltpdv_server_import</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_queryServer">m/gui/ltpdv/callbacks/ltpdv_server_queryServer</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_server_select">m/gui/ltpdv/callbacks/ltpdv_server_server_select</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_set_channel_list">m/gui/ltpdv/callbacks/ltpdv_server_set_channel_list</a>
- Get normal channels
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_set_duration">m/gui/ltpdv/callbacks/ltpdv_server_set_duration</a>
- Get handle
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_set_timeslist">m/gui/ltpdv/callbacks/ltpdv_server_set_timeslist</a>
- Get times list
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_start_edit">m/gui/ltpdv/callbacks/ltpdv_server_start_edit</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_stop_edit">m/gui/ltpdv/callbacks/ltpdv_server_stop_edit</a>
- Handles

```

```
m/gui/ltpdv/callbacks/ltpdv_server_timeslist          - (No help available)
<a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_set_filter_list">m/gui/ltpdv/callbacks/ltpdv_set_filter_list</a>
- Get filters
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_set_pole_list">m/gui/ltpdv/callbacks/ltpdv_set_pole_list</a>
- Get poles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_set_zero_list">m/gui/ltpdv/callbacks/ltpdv_set_zero_list</a>
- Get poles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_spectral_fcncall">m/gui/ltpdv/callbacks/ltpdv_spectral_fcncall</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_storeFilter">m/gui/ltpdv/callbacks/ltpdv_storeFilter</a>
- stores the current filter as an object in the object
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_storeFilterResp">m/gui/ltpdv/callbacks/ltpdv_storeFilterResp</a>
- stores the response on the current filter in the
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_tabChanged">m/gui/ltpdv/callbacks/ltpdv_tabChanged</a>
- get old tab
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_workspaceList_select">m/gui/ltpdv/callbacks/ltpdv_workspaceList_select</a>
- activates when the user selects cells in the
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_workspaceList_set_data">m/gui/ltpdv/callbacks/ltpdv_workspaceList_set_data</a>
- sets the data into the workspace table
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_zeroAdd">m/gui/ltpdv/callbacks/ltpdv_zeroAdd</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_zeroClear">m/gui/ltpdv/callbacks/ltpdv_zeroClear</a>
- Handles
  <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_zeroDelete">m/gui/ltpdv/callbacks/ltpdv_zeroDelete</a>
- Handles
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpdv_buildFilterPanel

PURPOSE ^

SYNOPSIS ^

```
function ltpdv_buildFilterPanel(panH, mainfig)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_filterAdd](#) Handles
-  [ltpdv_filterApply](#) Handles
-  [ltpdv_filterBrowse](#) Handles
-  [ltpdv_filterDelete](#) Handles
-  [ltpdv_filterImport](#) Handles
-  [ltpdv_filter_list_select](#) Handles
-  [ltpdv_filtersClear](#) Handles
-  [ltpdv_plotFilterResp](#) LTPDV_PLOTFILTERRESP plots the response of the selected filter.
-  [ltpdv_poleAdd](#) Handles
-  [ltpdv_poleClear](#) Handles
-  [ltpdv_poleDelete](#) Handles
-  [ltpdv_pzmResp](#) Handles
-  [ltpdv_pzmStore](#) Handles
-  [ltpdv_pzmStoreResp](#) Handles
-  [ltpdv_storeFilter](#) LTPDV_STOREFILTER stores the current filter as an object in the object
-  [ltpdv_storeFilterResp](#) LTPDV_STOREFILTERRESP stores the response on the current filter in the
-  [ltpdv_zeroAdd](#) Handles
-  [ltpdv_zeroClear](#) Handles
-  [ltpdv_zeroDelete](#) Handles

This function is called by:

SUBFUNCTIONS ^

-  [function buildFilterPoleZeroPanel\(panH, mainfig\)](#)
-  [function buildFilterObjectPanel\(panH, mainfig\)](#)
-  [function buildFilterFilePanel\(panH, mainfig\)](#)
-  [function buildFilterStandardPanel\(panH, mainfig\)](#)
-  [function addEntry\(panH, str, val, pmarg, pl, pb, pw, ph\)](#)

SOURCE CODE ^

```
0001 function ltpdv_buildFilterPanel(panH, mainfig)
0002
0003 Gproperties = getappdata(mainfig, 'Gproperties');
```



```

0004
0005 pmarg = 0.025;
0006 Hmargin = 0.05;
0007 Vmargin = 0.05;
0008
0009
0010
0011 %% Add new sub tabbed panel
0012
0013 Tleft = pmarg+0.3;
0014 Tbottom = 0.17;
0015 Theight = 1-pmarg - Tbottom;
0016 Twidth = 1-Tleft-pmarg;
0017
0018 panh = uipanel('Parent', panH, ...
0019     'BackgroundColor', [0.8 0.8 0.8], ...
0020     'Units', 'normalized', ...
0021     'Position', [Tleft Tbottom Twidth Theight]);
0022
0023 Tleft = pmarg;
0024 Tbottom = pmarg;
0025 Twidth = 1-2*pmarg;
0026 Theight = 1-2*pmarg;
0027 htab = uitabgroup('Parent', panh, ...
0028     'Units', 'normalized', ...
0029     'Position', [Tleft Tbottom Twidth Theight]);
0030 t1 = uitab(htab, 'title', 'Standard');
0031 t2 = uitab(htab, 'title', 'File');
0032 t3 = uitab(htab, 'title', 'Object');
0033 t4 = uitab(htab, 'title', 'Poles/Zeros');
0034
0035
0036 %----- Build Standard Panel
0037 buildFilterStandardPanel(t1, mainfig);
0038 %----- Build File Panel
0039 buildFilterFilePanel(t2, mainfig);
0040 %----- Build Object Panel
0041 buildFilterObjectPanel(t3, mainfig);
0042 %----- Build Poles/zeros Panel
0043 buildFilterPoleZeroPanel(t4, mainfig);
0044
0045 %% Filter list
0046
0047 bl = pmarg;
0048 bh = 0.025;
0049 bb = 1-pmarg-bh;
0050 bw = 0.3-2*pmarg;
0051 sth = uicontrol(panH, 'Style', 'text', ...
0052     'String', 'Filters', ...
0053     'Units', 'normalized', ...
0054     'BackgroundColor', 'w', ...
0055     'FontSize', 12, ...
0056     'Position', [bl bb bw bh]);
0057
0058 ll = pmarg;
0059 lh = 0.5;
0060 lb = bb-pmarg-lh;
0061 lw = 0.3-2*pmarg;
0062 lh = uicontrol(panH, 'Style', 'listbox', ...
0063     'String', ...
0064     'BackgroundColor', 'w', ...
0065     'Value', 1, 'Units', 'normalized', ...
0066     'min', 1, 'max', 1000, ...
0067     'FontSize', 14, ...
0068     'Tag', 'LTPDVfilterList', ...
0069     'Callback', {@ltpdv_filter_list_select, mainfig}, ...
0070     'Position', [ll lb lw lh]);
0071
0072 % Add filter button
0073 bl = pmarg;
0074 bh = 0.05;
0075 bb = lb - bh;
0076 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0077     'String', 'Add Filter', ...
0078     'Callback', {@ltpdv_filterAdd, htab, mainfig}, ...
0079     'Units', 'normalized', ...
0080     'Position', [bl bb bw bh]);
0081
0082 % Delete filter button
0083 bl = pmarg;
0084 bb = bb - bh;
0085 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0086     'String', 'Delete Filter', ...
0087     'Callback', {@ltpdv_filterDelete, htab, mainfig}, ...
0088     'Units', 'normalized', ...
0089     'Position', [bl bb bw bh]);

```

```

0090
0091 % Clear filter button
0092 bl = pmarg;
0093 bb = bb - bh;
0094 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0095     'String', 'Clear Filters', ...
0096     'Callback', {@ltpdv_filtersClear, htab, mainfig}, ...
0097     'Units', 'normalized', ...
0098     'Position', [bl bb bw bh]);
0099
0100 % Apply button
0101 bw = 0.15;
0102 bl = Hmargin;
0103 bb = 2*pmarg+bh;
0104 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0105     'String', 'Apply Filters', ...
0106     'Callback', {@ltpdv_filterApply, htab, mainfig}, ...
0107     'Units', 'normalized', ...
0108     'Position', [bl bb bw bh]);
0109
0110 % Store filter
0111 bl = Hmargin;
0112 bb = pmarg;
0113 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0114     'String', 'Store Filters', ...
0115     'Callback', {@ltpdv_storeFilter, htab, mainfig}, ...
0116     'Units', 'normalized', ...
0117     'Position', [bl bb bw bh]);
0118
0119 % Plot Filter response
0120 bl = bl + bw + pmarg;
0121 bb = 2*pmarg+bh;
0122 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0123     'String', 'Plot Filter Resps', ...
0124     'Callback', {@ltpdv_plotFilterResp, htab, mainfig}, ...
0125     'Units', 'normalized', ...
0126     'Position', [bl bb bw bh]);
0127
0128 % Store response
0129 bb = pmarg;
0130 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0131     'String', 'Store Filter Resps', ...
0132     'Callback', {@ltpdv_storeFilterResp, htab, mainfig}, ...
0133     'Units', 'normalized', ...
0134     'Position', [bl bb bw bh]);
0135
0136 % f1
0137 tw = 0.1;
0138 bl = bl + bw + pmarg;
0139 bw = 0.15;
0140 bb = 2*pmarg+bh;
0141 sth = uicontrol(panH, 'Style', 'text', ...
0142     'String', 'Start F', ...
0143     'Units', 'normalized', ...
0144     'BackgroundColor', 'w', ...
0145     'FontSize', 12, ...
0146     'Position', [bl bb tw bh]);
0147 bl = bl+tw+pmarg;
0148 sth = uicontrol(panH, 'Style', 'edit', ...
0149     'String', '', ...
0150     'Units', 'normalized', ...
0151     'BackgroundColor', 'w', ...
0152     'FontSize', 12, ...
0153     'Tag', 'LTPDVfilterF1', ...
0154     'Position', [bl bb bw bh]);
0155
0156
0157 % f2
0158 bb = pmarg;
0159 bl = bl -tw-pmarg;
0160 sth = uicontrol(panH, 'Style', 'text', ...
0161     'String', 'Stop F', ...
0162     'Units', 'normalized', ...
0163     'BackgroundColor', 'w', ...
0164     'FontSize', 12, ...
0165     'Position', [bl bb tw bh]);
0166 bl = bl+tw+pmarg;
0167 sth = uicontrol(panH, 'Style', 'edit', ...
0168     'String', '', ...
0169     'Units', 'normalized', ...
0170     'BackgroundColor', 'w', ...
0171     'FontSize', 12, ...
0172     'Tag', 'LTPDVfilterF2', ...
0173     'Position', [bl bb bw bh]);
0174
0175 % nf

```

```

0176 bl = bl + bw + pmarg;
0177 bb = 2*pmarg+bh;
0178 sth = uicontrol(panH, 'Style','text',...
0179     'String','Num f',...
0180     'Units','normalized',...
0181     'BackgroundColor','w',...
0182     'FontSize',12,...
0183     'Position',[bl bb tw bh]);
0184 bl = bl+tw+pmarg;
0185 sth = uicontrol(panH, 'Style','edit',...
0186     'String','',...
0187     'Units','normalized',...
0188     'BackgroundColor','w',...
0189     'FontSize',12,...
0190     'Tag','LTPDVfilterNf',...
0191     'Position',[bl bb bw bh]);
0192
0193 % scale
0194 bb = pmarg;
0195 bl = bl - tw - pmarg;
0196 sth = uicontrol(panH, 'Style','text',...
0197     'String','Scale',...
0198     'Units','normalized',...
0199     'BackgroundColor','w',...
0200     'FontSize',12,...
0201     'Position',[bl bb tw bh]);
0202 bl = bl+tw+pmarg;
0203 sth = uicontrol(panH, 'Style','popupmenu',...
0204     'String',{'log','lin'},...
0205     'Units','normalized',...
0206     'BackgroundColor','w',...
0207     'FontSize',12,...
0208     'Tag','LTPDVfilterScale',...
0209     'Position',[bl bb bw bh]);
0210
0211
0212 %-----
0213 % Build filter pole/zero panel
0214 function buildFilterPoleZeroPanel(panH, mainfig)
0215
0216     pmarg = 0.025;
0217
0218     % freq entry
0219     bl = pmarg;
0220     bw = 0.15;
0221     tw = 0.2;
0222     bh = 0.05;
0223     bb = 1 - bh - pmarg;
0224     sth = uicontrol(panH, 'Style','text',...
0225         'String','Freq [Hz]',...
0226         'Units','normalized',...
0227         'BackgroundColor','w',...
0228         'FontSize',12,...
0229         'Position',[bl bb tw bh]);
0230     bl = bl+tw+pmarg;
0231     sth = uicontrol(panH, 'Style','edit',...
0232         'String','',...
0233         'Units','normalized',...
0234         'BackgroundColor','w',...
0235         'FontSize',12,...
0236         'Tag','LTPDVfilterPZfreq',...
0237         'Position',[bl bb bw bh]);
0238
0239     % q entry
0240     bl = bl + bw + pmarg;
0241     bb = 1 - bh - pmarg;
0242     sth = uicontrol(panH, 'Style','text',...
0243         'String','Q',...
0244         'Units','normalized',...
0245         'BackgroundColor','w',...
0246         'FontSize',12,...
0247         'Position',[bl bb tw/2 bh]);
0248     bl = bl+tw+pmarg;
0249     sth = uicontrol(panH, 'Style','edit',...
0250         'String','',...
0251         'Units','normalized',...
0252         'BackgroundColor','w',...
0253         'FontSize',12,...
0254         'Tag','LTPDVfilterPZq',...
0255         'Position',[bl bb bw bh]);
0256
0257     %----- Poles
0258     pl = pmarg;
0259     pw = 0.5 - 2*pmarg;
0260     ph = 0.6;
0261     pb = bb - pmarg - ph;

```

```

0262     poleh = uipanel(panH, 'Title', 'Poles', ...
0263         'Units', 'normalized', ...
0264         'Position', [pl pb pw ph]);
0265 % add btn
0266 bl = pmarg;
0267 bh = 0.07;
0268 bb = 1-bh-pmarg;
0269 bw = 1-2*pmarg;
0270 pbh = uicontrol(poleh, 'Style', 'pushbutton', ...
0271     'String', 'Add', ...
0272     'Callback', {@ltpdv_poleAdd, htab, mainfig}, ...
0273     'Units', 'normalized', ...
0274     'Position', [bl bb bw bh]);
0275
0276 % list
0277 ll = pmarg;
0278 lh = 0.7;
0279 lb = bb-lh-pmarg;
0280 lw = 1-2*pmarg;
0281 lh = uicontrol(poleh, 'Style', 'listbox', ...
0282     'String', '', ...
0283     'BackgroundColor', 'w', ...
0284     'Value', 1, 'Units', 'normalized', ...
0285     'FontSize', 14, ...
0286     'Tag', 'LTPDVfilterPolelist', ...
0287     'max', 1000, ...
0288     'Position', [ll lb lw lh]);
0289
0290 % del btn
0291 bl = pmarg;
0292 bb = lb-pmarg-bh;
0293 pbh = uicontrol(poleh, 'Style', 'pushbutton', ...
0294     'String', 'Delete', ...
0295     'Callback', {@ltpdv_poleDelete, htab, mainfig}, ...
0296     'Units', 'normalized', ...
0297     'Position', [bl bb bw bh]);
0298 % clear btn
0299 bl = pmarg;
0300 bb = bb-bh;
0301 pbh = uicontrol(poleh, 'Style', 'pushbutton', ...
0302     'String', 'Clear', ...
0303     'Callback', {@ltpdv_poleClear, htab, mainfig}, ...
0304     'Units', 'normalized', ...
0305     'Position', [bl bb bw bh]);
0306
0307 %----- Zeros
0308 pl = 0.5 + pmarg;
0309 zeroh = uipanel(panH, 'Title', 'Zeros', ...
0310     'Units', 'normalized', ...
0311     'Position', [pl pb pw ph]);
0312 % Add
0313 bl = pmarg;
0314 bh = 0.07;
0315 bb = 1-bh-pmarg;
0316 bw = 1-2*pmarg;
0317 pbh = uicontrol(zeroh, 'Style', 'pushbutton', ...
0318     'String', 'Add', ...
0319     'Callback', {@ltpdv_zeroAdd, htab, mainfig}, ...
0320     'Units', 'normalized', ...
0321     'Position', [bl bb bw bh]);
0322 % list
0323 ll = pmarg;
0324 lh = 0.7;
0325 lb = bb-lh-pmarg;
0326 lw = 1-2*pmarg;
0327 lh = uicontrol(zeroh, 'Style', 'listbox', ...
0328     'String', '', ...
0329     'BackgroundColor', 'w', ...
0330     'Value', 1, 'Units', 'normalized', ...
0331     'FontSize', 14, ...
0332     'Tag', 'LTPDVfilterZerolist', ...
0333     'max', 1000, ...
0334     'Position', [ll lb lw lh]);
0335 % del btn
0336 bl = pmarg;
0337 bb = lb-pmarg-bh;
0338 pbh = uicontrol(zeroh, 'Style', 'pushbutton', ...
0339     'String', 'Delete', ...
0340     'Callback', {@ltpdv_zeroDelete, htab, mainfig}, ...
0341     'Units', 'normalized', ...
0342     'Position', [bl bb bw bh]);
0343 % clear btn
0344 bl = pmarg;
0345 bb = bb-bh;
0346 pbh = uicontrol(zeroh, 'Style', 'pushbutton', ...
0347     'String', 'Clear', ...

```

```

0348         'Callback', {@ltpdv_zeroClear, htab, mainfig}, ...
0349         'Units', 'normalized', ...
0350         'Position', [bl bb bw bh]);
0351
0352 % Gain entry
0353 pos = get(zeroH, 'Position');
0354 bl = pmarg;
0355 bw = 0.15;
0356 tw = 0.15;
0357 bh = 0.06;
0358 bb = pos(2) - bh - pmarg;
0359 sth = uicontrol(panH, 'Style', 'text', ...
0360     'String', 'Gain', ...
0361     'Units', 'normalized', ...
0362     'BackgroundColor', 'w', ...
0363     'FontSize', 12, ...
0364     'Position', [bl bb tw bh]);
0365 bl = bl+tw+pmarg;
0366 sth = uicontrol(panH, 'Style', 'edit', ...
0367     'String', '', ...
0368     'Units', 'normalized', ...
0369     'BackgroundColor', 'w', ...
0370     'FontSize', 12, ...
0371     'Tag', 'LTPDVfilterPZgain', ...
0372     'Position', [bl bb bw bh]);
0373
0374 % Name entry
0375 bl = bl + bw + pmarg;
0376 sth = uicontrol(panH, 'Style', 'text', ...
0377     'String', 'Name', ...
0378     'Units', 'normalized', ...
0379     'BackgroundColor', 'w', ...
0380     'FontSize', 12, ...
0381     'Position', [bl bb tw bh]);
0382 bl = bl+tw+pmarg;
0383 bw = 0.3;
0384 sth = uicontrol(panH, 'Style', 'edit', ...
0385     'String', '', ...
0386     'Units', 'normalized', ...
0387     'BackgroundColor', 'w', ...
0388     'FontSize', 12, ...
0389     'Tag', 'LTPDVfilterPZname', ...
0390     'Position', [bl bb bw bh]);
0391
0392 % Pole/Zero resp button
0393 bb = bb - bh - pmarg;
0394 bl = pmarg;
0395 bw = 0.22;
0396 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0397     'String', 'Plot response', ...
0398     'Callback', {@ltpdv_pzmResp, panH, mainfig}, ...
0399     'Units', 'normalized', ...
0400     'Position', [bl bb bw bh]);
0401
0402 % Pole/Zero store button
0403 bl = bl+bw+pmarg;
0404 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0405     'String', 'Store model', ...
0406     'Callback', {@ltpdv_pzmStore, panH, mainfig}, ...
0407     'Units', 'normalized', ...
0408     'Position', [bl bb bw bh]);
0409
0410 % Pole/Zero store resp button
0411 bl = bl+bw+pmarg;
0412 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0413     'String', 'Store model resp', ...
0414     'Callback', {@ltpdv_pzmStoreResp, panH, mainfig}, ...
0415     'Units', 'normalized', ...
0416     'Position', [bl bb bw bh]);
0417
0418 end
0419
0420 %-----
0421 % Build filter object panel
0422 function buildFilterObjectPanel(panH, mainfig)
0423
0424     pmarg = 0.025;
0425
0426 % import button
0427 bl = pmarg;
0428 bh = 0.04;
0429 bw = 0.2;
0430 bb = 1 - bh - pmarg;
0431 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0432     'String', 'Import Filter', ...
0433     'Callback', {@ltpdv_filterImport, panH, mainfig}, ...

```

```

0434         'Units', 'normalized', ...
0435         'Position', [bl bb bw bh]);
0436
0437 % info panel
0438 % Filter info panel
0439 tl = pmarg;
0440 tw = 1 - 2*pmarg;
0441 th = bb - 2*pmarg;
0442 tb = pmarg;
0443 ph = uipanel(panH, 'Title', 'Imported Filter', ...
0444             'Units', 'normalized', ...
0445             'FontSize', 12, ...
0446             'Position', [tl tb tw th]);
0447
0448 tl = pmarg;
0449 tb = pmarg;
0450 tw = 1 - 2*pmarg;
0451 th = 1 - 2*pmarg;
0452 fh = uicontrol(ph, 'Style', 'text', ...
0453             'String', '', ...
0454             'HorizontalAlignment', 'left', ...
0455             'Units', 'normalized', ...
0456             'FontSize', 12, ...
0457             'FontName', 'monospace', ...
0458             'BackgroundColor', 'w', ...
0459             'Tag', 'LTPDVfilterObjectInfo', ...
0460             'Position', [tl tb tw th]);
0461
0462 end
0463 %-----
0464 % Build file filters panel
0465 function buildFilterFilePanel(panH, mainfig)
0466
0467     pmarg = 0.025;
0468
0469 % Text label
0470 tl = pmarg;
0471 th = 0.05;
0472 tw = 0.15;
0473 tb = 1 - pmarg - th;
0474 sth = uicontrol(panH, 'Style', 'text', ...
0475             'String', 'Filename', ...
0476             'Units', 'normalized', ...
0477             'BackgroundColor', 'w', ...
0478             'FontSize', 12, ...
0479             'Position', [tl tb tw th]);
0480
0481 % File entry field
0482 tl = tl + tw + pmarg;
0483 th = 0.05;
0484 tw = 0.5;
0485 tb = 1 - pmarg - th;
0486 sth = uicontrol(panH, 'Style', 'edit', ...
0487             'String', '', ...
0488             'Units', 'normalized', ...
0489             'BackgroundColor', 'w', ...
0490             'FontSize', 12, ...
0491             'Tag', 'LTPDVfilterFileFilename', ...
0492             'Position', [tl tb tw th]);
0493
0494 % Browse button
0495 tl = tl + tw + pmarg;
0496 th = 0.05;
0497 tw = 0.2;
0498 tb = 1 - pmarg - th;
0499 sth = uicontrol(panH, 'Style', 'pushbutton', ...
0500             'String', 'Browse', ...
0501             'Units', 'normalized', ...
0502             'FontSize', 12, ...
0503             'Callback', {@ltpdv_filterBrowse, panH, mainfig}, ...
0504             'Position', [tl tb tw th]);
0505
0506 % Filter info panel
0507 tl = pmarg;
0508 tw = 1 - 2*pmarg;
0509 th = tb - 2*pmarg;
0510 tb = pmarg;
0511 ph = uipanel(panH, 'Title', 'Loaded Filter', ...
0512             'Units', 'normalized', ...
0513             'FontSize', 12, ...
0514             'Position', [tl tb tw th]);
0515
0516 tl = pmarg;
0517 tb = pmarg;
0518 tw = 1 - 2*pmarg;
0519 th = 1 - 2*pmarg;

```



```

0520     fh = uicontrol(ph, 'Style', 'text', ...
0521     'String', '', ...
0522     'HorizontalAlignment', 'left', ...
0523     'Units', 'normalized', ...
0524     'FontSize', 12, ...
0525     'FontName', 'monospace', ...
0526     'BackgroundColor', 'w', ...
0527     'Tag', 'LTPDVfilterFileInfo', ...
0528     'Position', [tl tb tw th]);
0529
0530
0531
0532 end
0533 %-----
0534 % Build standard filters panel
0535 function buildFilterStandardPanel(panH, mainfig)
0536
0537     filts = {'Lowpass', 'Highpass', 'Bandpass', 'Bandreject'};
0538
0539     pmarg = 0.025;
0540
0541
0542     % Filter selection box
0543     pl = pmarg;
0544     ph = 0.05;
0545     pb = 1 - ph - 2*pmarg;
0546     pw = 0.4;
0547     eh = uicontrol(panH, 'Style', 'popupmenu', ...
0548     'String', filts, ...
0549     'units', 'normalized', ...
0550     'BackgroundColor', 'w', ...
0551     'FontSize', 12, ...
0552     'Enable', 'on', ...
0553     'Tag', 'LTPDVfiltStandardType', ...
0554     'Position', [pl pb pw ph]);
0555
0556     % Gain
0557     addEntry(panH, 'Gain', 1, pmarg, pl, pb, pw, ph);
0558     % Fc
0559     pb = pb - ph - pmarg;
0560     addEntry(panH, 'Cutoff', 0.1, pmarg, pl, pb, pw, ph);
0561     % fs
0562     pb = pb - ph - pmarg;
0563     addEntry(panH, 'Fs', 1, pmarg, pl, pb, pw, ph);
0564     % order
0565     pb = pb - ph - pmarg;
0566     addEntry(panH, 'Order', 1, pmarg, pl, pb, pw, ph);
0567     % ripple
0568     pb = pb - ph - pmarg;
0569     addEntry(panH, 'Ripple', 0.5, pmarg, pl, pb, pw, ph);
0570
0571
0572
0573 end
0574
0575 function addEntry(panH, str, val, pmarg, pl, pb, pw, ph)
0576
0577     pl = pmarg;
0578     pb = pb - ph - pmarg;
0579     sth = uicontrol(panH, 'Style', 'text', ...
0580     'String', str, ...
0581     'Units', 'normalized', ...
0582     'BackgroundColor', 'w', ...
0583     'FontSize', 12, ...
0584     'Position', [pl pb pw ph]);
0585     pl = 2*pmarg + pw;
0586     eh = uicontrol(panH, 'Style', 'edit', ...
0587     'String', val, ...
0588     'units', 'normalized', ...
0589     'BackgroundColor', 'w', ...
0590     'FontSize', 12, ...
0591     'Enable', 'on', ...
0592     'Tag', sprintf('LTPDVfiltStandard%s', str), ...
0593     'Position', [pl pb pw ph]);
0594
0595 end
0596
0597 end

```

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_buildImportPanel.m

ltpdv_buildImportPanel

PURPOSE [^]

SYNOPSIS [^]

```
function ltpdv_buildImportPanel(ph, mainfig)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_build_buildobject](#) LTPDV_BUILD_BUILDOBJECT callback for the build object button on the build
-  [ltpdv_build_class_select](#) LTPDV_BUILD_CLASS_SELECT callback when the user selects the class
-  [ltpdv_build_set_select](#) LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to
-  [ltpdv_importObjs](#) LTPDV_IMPORTOBJs import given objects to the object list
-  [ltpdv_search_channels](#) LTPDV_SEARCH_CHANNELS search the channel list and reduce it to those that
-  [ltpdv_server_addTime](#) Handles
-  [ltpdv_server_channellist](#) Handles
-  [ltpdv_server_clearTimes](#) Handles
-  [ltpdv_server_datatype_select](#) Handles
-  [ltpdv_server_deleteTimes](#) Handles
-  [ltpdv_server_getChannels](#)
-  [ltpdv_server_getLatest](#) Handles
-  [ltpdv_server_import](#) Handles
-  [ltpdv_server_queryServer](#) Handles
-  [ltpdv_server_server_select](#) Handles
-  [ltpdv_server_start_edit](#) Handles
-  [ltpdv_server_stop_edit](#) Handles
-  [ltpdv_server_timeslist](#)
-  [ltpdv_tabChanged](#)
-  [ltpdv_workspaceList_select](#) LTPDV_WORKSPACELIST_SELECT activates when the user selects cells in the
-  [ltpdv_workspaceList_set_data](#) LTPDV_WORKSPACELIST_SET_DATA sets the data into the workspace table

This function is called by:

SUBFUNCTIONS [^]

-  [function buildBuildPanel](#)(hp, mainfig)
-  [function buildServerPanel](#)(hp, mainfig)
-  [function th = buildWorkspacePanel](#)(hp, mainfig)

SOURCE CODE ^

```

0001 function ltpdv_buildImportPanel(ph, mainfig)
0002
0003     Gproperties = getappdata(mainfig, 'Gproperties');
0004
0005     pmarg = 0.025;
0006     % Build a panel to contain this tab
0007     % panh = uipanel('Parent', ph, 'Position', [pmarg pmarg 1-2*pmarg 1-2*pmarg]);
0008
0009     %% Add new sub tabbed panel
0010
0011     Hmargin = 0.025;
0012     Vmargin = 0.025;
0013
0014     PanLeft = Hmargin;
0015     Tleft = Hmargin;
0016     Theight = 1-2*Vmargin;
0017     Tbottom = (1 - Theight - Vmargin);
0018     Twidth = 1-2*Hmargin;
0019
0020     Pcol = [1 1 0.9];
0021
0022     htab = uitabgroup('Parent', ph, ...
0023         'Units', 'normalized', ...
0024         'Tag', 'LTPDVimportTabPanel', ...
0025         'Position', [Tleft Tbottom Twidth Theight], 'SelectionChangeFcn',
0026         {@ltpdv_tabChanged});
0027     mt1 = uitab(htab, 'title', 'Server');
0028     mt2 = uitab(htab, 'title', 'Workspace');
0029     mt3 = uitab(htab, 'title', 'Build');
0030
0031     %----- Build Server Panel
0032     buildServerPanel(mt1, mainfig);
0033     %----- Build Workspace Panel
0034     h = buildWorkspacePanel(mt2, mainfig);
0035     %----- Build Build Panel
0036     buildBuildPanel(mt3, mainfig);
0037 end
0038 %-----
0039 % Build import from server panel
0040 function buildBuildPanel(hp, mainfig)
0041
0042     EditFontSize = 11;
0043     vmarg = 0.03;
0044     pmarg = 0.01;
0045     CommHeight = EditFontSize/400;
0046     ph = 1;
0047     htab = uipanel('Parent', hp, 'Position', [pmarg 1-ph-pmarg 1-2*pmarg ph]);
0048     set(htab, 'Units', 'Normalized')
0049     set(htab, 'Title', '')
0050     set(htab, 'BackgroundColor', [0.6 0.6 0.8]);
0051     set(htab, 'ForegroundColor', 'w');
0052
0053     %----- Class selector
0054     bh = CommHeight;
0055     bl = pmarg;
0056     bw = 0.15;
0057     bb = 1-bh-1*vmarg;
0058     sth = uicontrol(htab, 'Style', 'text', ...
0059         'String', 'Object Class', ...
0060         'Units', 'normalized', ...
0061         'BackgroundColor', 'w', ...
0062         'FontSize', EditFontSize, ...
0063         'Position', [bl bb bw bh]);
0064     bl = bl+bw+pmarg;
0065     bw = 0.2;
0066     clsh = uicontrol(htab, ...
0067         'Style', 'popupmenu', ...
0068         'Units', 'normalized', ...
0069         'FontSize', EditFontSize, ...
0070         'BackgroundColor', 'w', ...
0071         'String', utils.helper.ltpda_userclasses, ...
0072         'Position', [bl bb bw bh], ...
0073         'Tag', 'LTPDVbuildObjectClass', ...
0074         'Callback', {@ltpdv_build_class_select, mainfig});
0075
0076     %----- Set selector
0077     bl = bl+bw+pmarg;
0078     bw = 0.15;
0079     sth = uicontrol(htab, 'Style', 'text', ...

```

```

0082     'String', 'Parameter Set', ...
0083     'Units', 'normalized', ...
0084     'BackgroundColor', 'w', ...
0085     'FontSize', EditFontSize, ...
0086     'Position', [bl bb bw bh]);
0087     bl = bl+bw+pmarg;
0088     bw = 0.3;
0089     dtypeh = uicontrol(htab, ...
0090     'Style', 'popupmenu', ...
0091     'Units', 'normalized', ...
0092     'FontSize', EditFontSize, ...
0093     'BackgroundColor', 'w', ...
0094     'String', '...', ...
0095     'Position', [bl bb bw bh], ...
0096     'Tag', 'LTPDVbuildSetSelect', ...
0097     'Callback', {@ltpdv_build_set_select, mainfig});
0098
0099 % Parameter panel
0100 ph = 0.8;
0101 ppan = uipanel('Parent', htab, 'Position', [pmarg 0.08+vmarg 1-2*pmarg ph]);
0102 set(ppan, 'Units', 'Normalized')
0103 set(ppan, 'Title', 'Parameters')
0104 set(ppan, 'BackgroundColor', [0.7 0.7 0.8]);
0105 set(ppan, 'ForegroundColor', 'w');
0106 set(ppan, 'Tag', 'LTPDVbuildParamsPanel');
0107
0108 pbh = uicontrol(htab, 'Style', 'pushbutton', ...
0109 'String', 'Build object', ...
0110 'Callback', {@ltpdv_build_buildobject, mainfig}, ...
0111 'Units', 'normalized', ...
0112 'FontSize', EditFontSize, ...
0113 'Position', [pmarg vmarg 0.15 2*bh]);
0114
0115 % call back
0116 ltpdv_build_class_select(clsh, mainfig)
0117
0118 %
0119 %     bh = CommHeight;
0120 %     bl = pmarg;
0121 %     bw = 1-2*pmarg;
0122 %     bb = vmarg;
0123 %     sth = uicontrol(hp, 'Style','text',...
0124 %     'String', 'Build objects with the ''constructor helper'' then import from
workspace.', ...
0125 %     'Units', 'normalized', ...
0126 %     'BackgroundColor', 'w', ...
0127 %     'FontSize', EditFontSize, ...
0128 %     'Position', [bl bb bw bh]);
0129
0130 end
0131 %-----
0132 % Build import from server panel
0133 function buildServerPanel(hp, mainfig)
0134
0135 % Data type
0136 EditFontSize = 11;
0137 vmarg = 0.03;
0138 pmarg = 0.01;
0139 CommHeight = EditFontSize/80;
0140
0141 % Import button
0142 bw = 0.12;
0143 bh = 0.05;
0144 bl = pmarg;
0145 bb = pmarg;
0146 pbh = uicontrol(hp, 'Style', 'pushbutton', ...
0147 'String', 'Import', ...
0148 'Callback', {@ltpdv_server_import, mainfig}, ...
0149 'Units', 'normalized', ...
0150 'FontSize', EditFontSize, ...
0151 'Position', [bl bb bw bh]);
0152
0153 %===== New tab panel
0154 Tleft = pmarg;
0155 Theight = 0.8;
0156 Tbottom = 1-Theight - pmarg;
0157 Twidth = 1-2*pmarg;
0158 %     htab = uitabgroup('Parent', hp, ...
0159 %     'Units','normalized',...
0160 %     'Position',[Tleft Tbottom Twidth Theight], 'SelectionChangeFcn',
{@ltpdv_tabChanged});
0161 %     t1 = uitab(htab, 'title', 'Server Details');
0162 %     t2 = uitab(htab, 'title', 'Times');
0163 %     t3 = uitab(htab, 'title', 'Channels');
0164
0165 % ===== Server Details

```

```

0166
0167 ph = 0.3;
0168 htab = uipanel('Parent', hp, 'Position', [pmarg 1-ph-pmarg 1-2*pmarg ph]);
0169 set(htab, 'Units', 'Normalized')
0170 set(htab, 'Title', 'Server settings')
0171 set(htab, 'BackgroundColor', [0.3 0.7 0.3]);
0172 set(htab, 'ForegroundColor', 'w');
0173
0174 bh = CommHeight;
0175 bl = pmarg;
0176 bw = 0.1;
0177 bb = 1-bh-3*vmarg;
0178 sth = uicontrol(htab, 'Style','text',...
0179 'String', 'Data type',...
0180 'Units', 'normalized',...
0181 'BackgroundColor', 'w',...
0182 'FontSize', EditFontSize,...
0183 'Position',[bl bb bw bh]);
0184 bl = bl+bw+pmarg;
0185 bw = 0.17;
0186 dtypeh = uicontrol(htab, ...
0187 'Style', 'popupmenu', ...
0188 'Units', 'normalized',...
0189 'FontSize', EditFontSize, ...
0190 'BackgroundColor', 'w',...
0191 'String', {'frame data', 'h(t) frames', 'raw data', 'hour trends', 'day trends'},
...
0192 'Position', [bl bb bw bh],...
0193 'Tag', 'LTPDVserverDataType',...
0194 'Callback', {@ltpdv_server_datatype_select, mainfig});
0195
0196 % Statistic
0197 bl = bl + bw + pmarg;
0198 bw = 0.1;
0199 sth = uicontrol(htab, 'Style','text',...
0200 'String', 'Statistic',...
0201 'Units', 'normalized',...
0202 'BackgroundColor', 'w',...
0203 'FontSize', EditFontSize,...
0204 'Position',[bl bb bw bh]);
0205 bl = bl+bw+pmarg;
0206 bw = 0.12;
0207 th = uicontrol(htab, ...
0208 'Style', 'popupmenu', ...
0209 'Units', 'normalized',...
0210 'BackgroundColor', 'w',...
0211 'FontSize', EditFontSize,...
0212 'String', {'rms', 'min', 'max', 'avg'}, ...
0213 'Position', [bl bb bw bh],...
0214 'Tag', 'LTPDVserverStat');
0215
0216 % Server drop-down box
0217 bl = pmarg;
0218 bw = 0.1;
0219 bb = bb - bh - vmarg;
0220 sth = uicontrol(htab, 'Style','text',...
0221 'String', 'Server',...
0222 'Units', 'normalized',...
0223 'BackgroundColor', 'w',...
0224 'FontSize', EditFontSize,...
0225 'Position',[bl bb bw bh]);
0226 bl = bl+bw+pmarg;
0227 bw = 0.25;
0228 th = uicontrol(htab, ...
0229 'Style', 'popupmenu', ...
0230 'Units', 'normalized',...
0231 'BackgroundColor', 'w',...
0232 'String', {'130.75.117.159', '130.75.117.73', '130.75.117.164', 'localhost'}, ...
0233 'FontSize', EditFontSize,...
0234 'Position', [bl bb bw bh],...
0235 'Tag', 'LTPDVserverServer',...
0236 'Callback', {@ltpdv_server_server_select, mainfig});
0237
0238 % Server entry box
0239 bl = bl+bw+pmarg;
0240 bw = 0.22;
0241 th = uicontrol(htab, ...
0242 'Style', 'edit',...
0243 'Units', 'normalized',...
0244 'BackgroundColor', 'w',...
0245 'FontSize', EditFontSize,...
0246 'String', '',...
0247 'Position', [bl bb bw bh],...
0248 'Tag', 'LTPDVserverServerEdit');
0249
0250 % Port entry

```

```

0251 bl = bl+bw+pmarg;
0252 bw = 0.07;
0253 sth = uicontrol(htab, 'Style','text',...
0254     'String','Port',...
0255     'Units','normalized',...
0256     'BackgroundColor','w',...
0257     'FontSize', EditFontSize, ...
0258     'Position',[bl bb bw bh]);
0259 bl = bl+bw+pmarg;
0260 bw = 0.1;
0261 th = uicontrol(htab, ...
0262     'Style','edit',...
0263     'Units','normalized',...
0264     'BackgroundColor','w',...
0265     'FontSize', EditFontSize, ...
0266     'String','',...
0267     'Position',[bl bb bw bh],...
0268     'Tag','LTPDVserverPortEdit');
0269
0270 % RDS
0271 bl = pmarg;
0272 bw = 0.1;
0273 bb = bb - bh - vmarg;
0274 sth = uicontrol(htab, 'Style','text',...
0275     'String','RDS Level',...
0276     'Units','normalized',...
0277     'BackgroundColor','w',...
0278     'FontSize', EditFontSize, ...
0279     'Position',[bl bb bw bh]);
0280 bl = bl+bw+pmarg;
0281 bw = 0.05;
0282 th = uicontrol(htab, ...
0283     'Style','edit',...
0284     'Units','normalized',...
0285     'BackgroundColor','w',...
0286     'FontSize', EditFontSize, ...
0287     'String','',...
0288     'Position',[bl bb bw bh],...
0289     'Tag','LTPDVserverRdsEdit');
0290
0291 % CAL
0292 bl = pmarg + bl + bw;
0293 bw = 0.1;
0294 sth = uicontrol(htab, 'Style','text',...
0295     'String','Cal Ver',...
0296     'Units','normalized',...
0297     'BackgroundColor','w',...
0298     'FontSize', EditFontSize, ...
0299     'Position',[bl bb bw bh]);
0300 bl = bl+bw+pmarg;
0301 bw = 0.05;
0302 th = uicontrol(htab, ...
0303     'Style','edit',...
0304     'Units','normalized',...
0305     'BackgroundColor','w',...
0306     'FontSize', EditFontSize, ...
0307     'String','',...
0308     'Position',[bl bb bw bh],...
0309     'Tag','LTPDVserverCalEdit');
0310
0311
0312 % call callbacks
0313 ltpdv_server_server_select(0, mainfig);
0314 ltpdv_server_datatype_select(dtypeh, mainfig);
0315
0316 % Query server button
0317 bw = 0.13;
0318 bb = vmarg;
0319 bl = pmarg;
0320 pbh = uicontrol(htab, 'Style','pushbutton',...
0321     'String','Query Server',...
0322     'Callback', {@ltpdv_server_queryServer, mainfig}, ...
0323     'Units','normalized',...
0324     'FontSize', EditFontSize, ...
0325     'Position',[bl bb bw 1.4*CommHeight]);
0326
0327 % Get latest button
0328 bl = bl + bw + pmarg;
0329 pbh = uicontrol(htab, 'Style','pushbutton',...
0330     'String','Get latest',...
0331     'Callback', {@ltpdv_server_getLatest, mainfig}, ...
0332     'Units','normalized',...
0333     'FontSize', EditFontSize, ...
0334     'Position',[bl bb bw 1.4*CommHeight]);
0335
0336

```

```

0337 %----- Time entry
0338
0339 ph = 0.3;
0340 htab = uipanel('Parent', hp, 'Position', [pmarg 1-2*ph-2*pmarg 1-2*pmarg ph]);
0341 set(htab, 'Units', 'Normalized')
0342 set(htab, 'Title', 'Times settings')
0343 set(htab, 'BackgroundColor', [0.7 0.3 0.3]);
0344 set(htab, 'ForegroundColor', 'w');
0345
0346 % time and channel lists
0347 bb = 1-0.7*bh-pmarg;
0348 bl = pmarg+0.4;
0349 bw = 0.5;
0350 timesH = uicontrol(htab, 'Style','text',...
0351     'String','Times',...
0352     'Units','normalized',...
0353     'BackgroundColor','w',...
0354     'FontSize', EditFontSize,...
0355     'Position',[bl bb bw 0.7*bh]);
0356
0357 bl = pmarg + 0.4;
0358 bh = 0.7;
0359 bb = bb - bh;
0360 lh = uicontrol(htab, 'Style', 'listbox', ...
0361     'String',' ',...
0362     'BackgroundColor','w',...
0363     'Value',1, 'Units','normalized',...
0364     'FontSize', EditFontSize,...
0365     'Max', 1000,...
0366     'Tag', 'LTPDVtimesList',...
0367     'Position',[bl bb bw bh],...
0368     'Callback', {@ltpdv_server_timeslist, mainfig});
0369
0370 % delete times
0371 bw = (0.5 - 3*pmarg)/ 4;
0372 bh = CommHeight;
0373 bb = bb - bh - vmarg;
0374 pbh = uicontrol(htab, 'Style','pushbutton',...
0375     'String','Delete times',...
0376     'Callback', {@ltpdv_server_deleteTimes, mainfig},...
0377     'Units','normalized',...
0378     'FontSize', EditFontSize,...
0379     'Position',[bl bb bw bh]);
0380
0381 % clear list
0382 bl = bl + pmarg + bw;
0383 pbh = uicontrol(htab, 'Style','pushbutton',...
0384     'String','Clear list',...
0385     'Callback', {@ltpdv_server_clearTimes, mainfig},...
0386     'Units','normalized',...
0387     'FontSize', EditFontSize,...
0388     'Position',[bl bb bw bh]);
0389
0390 % save list
0391 bl = bl + pmarg + bw;
0392 pbh = uicontrol(htab, 'Style','pushbutton',...
0393     'String','Save list',...
0394     'Callback', {@ltpdv_server_saveTimes, mainfig},...
0395     'Units','normalized',...
0396     'FontSize', EditFontSize,...
0397     'Position',[bl bb bw bh]);
0398
0399 % load list
0400 bl = bl + pmarg + bw;
0401 pbh = uicontrol(htab, 'Style','pushbutton',...
0402     'String','Load list',...
0403     'Callback', {@ltpdv_server_loadTimes, mainfig},...
0404     'Units','normalized',...
0405     'FontSize', EditFontSize,...
0406     'Position',[bl bb bw bh]);
0407
0408 % start time
0409 pos = get(timesH, 'Position');
0410 bl = pmarg;
0411 bw = 0.1;
0412 bh = CommHeight;
0413 bb = pos(2)-2*vmarg;
0414 sth = uicontrol(htab, 'Style','text',...
0415     'String','Start',...
0416     'Units','normalized',...
0417     'BackgroundColor','w',...
0418     'FontSize', EditFontSize,...
0419     'Position',[bl bb bw bh]);
0420
0421 bl = bl+bw+pmarg;
0422 bw = 0.25;
0423 th = uicontrol(htab, ...
0424     'Style','edit',...
0425     'Units','normalized',...
0426     'BackgroundColor','w',...

```

```

0423     'FontSize', EditFontSize, ...
0424     'String', '', ...
0425     'Position', [bl bb bw bh], ...
0426     'Callback', {@ltpdv_server_start_edit, mainfig}, ...
0427     'Tag', 'LTPDVserverStartEdit');
0428 % stop time
0429 bl = pmarg;
0430 bw = 0.1;
0431 bb = bb - bh - vmarg;
0432 sth = uicontrol(htab, 'Style', 'text', ...
0433     'String', 'Stop', ...
0434     'Units', 'normalized', ...
0435     'BackgroundColor', 'w', ...
0436     'FontSize', EditFontSize, ...
0437     'Position', [bl bb bw bh]);
0438 bl = bl + bw + pmarg;
0439 bw = 0.25;
0440 th = uicontrol(htab, ...
0441     'Style', 'edit', ...
0442     'Units', 'normalized', ...
0443     'BackgroundColor', 'w', ...
0444     'String', '', ...
0445     'FontSize', EditFontSize, ...
0446     'Position', [bl bb bw bh], ...
0447     'Callback', {@ltpdv_server_stop_edit, mainfig}, ...
0448     'Tag', 'LTPDVserverStopEdit');
0449
0450 % duration display
0451 bw = 0.1;
0452 bl = pmarg;
0453 bb = bb - bh - vmarg;
0454 sth = uicontrol(htab, 'Style', 'text', ...
0455     'String', 'Duration', ...
0456     'Units', 'normalized', ...
0457     'BackgroundColor', 'w', ...
0458     'FontSize', EditFontSize, ...
0459     'Position', [bl bb bw bh]);
0460 bl = bl + bw + pmarg;
0461 bw = 0.25;
0462 sth = uicontrol(htab, 'Style', 'text', ...
0463     'String', '', ...
0464     'Units', 'normalized', ...
0465     'BackgroundColor', 'w', ...
0466     'FontSize', EditFontSize, ...
0467     'Tag', 'LTPDVserverDurationDisplay', ...
0468     'Position', [bl bb bw bh]);
0469
0470 % comment
0471 pos = get(sth, 'Position');
0472 bl = pmarg;
0473 bw = pos(1) + pos(3) - pmarg;
0474 bb = bb - 0.7*bh - vmarg;
0475 sth = uicontrol(htab, 'Style', 'text', ...
0476     'String', 'Comment', ...
0477     'Units', 'normalized', ...
0478     'BackgroundColor', 'w', ...
0479     'FontSize', EditFontSize, ...
0480     'Position', [bl bb bw bh*0.7]);
0481 bb = bb - bh - pmarg;
0482 bl = pmarg;
0483 th = uicontrol(htab, ...
0484     'Style', 'edit', ...
0485     'Units', 'normalized', ...
0486     'BackgroundColor', 'w', ...
0487     'String', '', ...
0488     'FontSize', EditFontSize, ...
0489     'Position', [bl bb bw bh], ...
0490     'Tag', 'LTPDVserverCommentEdit');
0491
0492 % Add time button
0493 bl = pmarg;
0494 bb = bb - bh - vmarg;
0495 pbh = uicontrol(htab, 'Style', 'pushbutton', ...
0496     'String', 'Add to list', ...
0497     'FontSize', EditFontSize, ...
0498     'Callback', {@ltpdv_server_addTime, mainfig}, ...
0499     'Units', 'normalized', ...
0500     'Position', [bl bb bw bh]);
0501
0502
0503 % ----- Channel list
0504 ph = 0.3;
0505 htab = uipanel('Parent', hp, 'Position', [pmarg pmarg+2*vmarg 1-2*pmarg ph]);
0506 set(htab, 'Units', 'Normalized')
0507 set(htab, 'Title', 'Channel settings')
0508 set(htab, 'BackgroundColor', [0.3 0.3 0.7]);

```



```

0509 set(htab, 'ForegroundColor', 'w');
0510
0511 bw = 0.5;
0512 bb = 1 - vmarg - 0.7*bh;
0513 bl = pmarg;
0514 sth = uicontrol(htab, 'Style','text',...
0515     'String','Channels',...
0516     'Units','normalized',...
0517     'BackgroundColor','w',...
0518     'FontSize', EditFontSize,...
0519     'Position',[bl bb bw 0.7*bh]);
0520 bh = 0.8;
0521 bb = bb - bh;
0522 lh = uicontrol(htab, 'Style', 'listbox', ...
0523     'String','...',...
0524     'BackgroundColor','w',...
0525     'Value',1, 'Units','normalized', ...
0526     'FontSize', EditFontSize, ...
0527     'Max', 1000, ...
0528     'Tag', 'LTPDVchannelList', ...
0529     'Position',[bl bb bw bh], ...
0530     'Callback', {@ltpdv_server_channellist, mainfig});
0531
0532 % include control chans
0533 %     bw = 0.3;
0534 %     bh = CommHeight;
0535 %     bb = bb - bh - pmarg;
0536 %     th = uicontrol(htab, ...
0537 %         'Style','checkbox',...
0538 %         'Units','normalized',...
0539 %         'BackgroundColor','w',...
0540 %         'String','Include control', ...
0541 %         'FontSize', EditFontSize, ...
0542 %         'Position',[bl bb bw bh],...
0543 %         'ForegroundColor','w',...
0544 %         'Tag','LTPDVchannelIncludeControl');
0545
0546 % search box
0547 bl = 0.55;
0548 bw = 0.08;
0549 bh = CommHeight;
0550 bb = 1 - vmarg - bh;
0551 sth = uicontrol(htab, 'Style','text',...
0552     'String','Search',...
0553     'Units','normalized',...
0554     'BackgroundColor','w',...
0555     'FontSize', EditFontSize, ...
0556     'Position',[bl bb bw bh]);
0557 bl = bl+bw+pmarg;
0558 bw = 0.17;
0559 th = uicontrol(htab, ...
0560     'Style','edit',...
0561     'Units','normalized',...
0562     'BackgroundColor','w',...
0563     'String','...',...
0564     'FontSize', EditFontSize, ...
0565     'Position',[bl bb bw bh],...
0566     'Callback', {@ltpdv_search_channels, mainfig}, ...
0567     'Tag','LTPDVchannelSearchEdit');
0568 pos = get(sth, 'Position');
0569 bl = pos(1);
0570 bb = bb - bh - vmarg;
0571 pbh = uicontrol(htab, 'Style','pushbutton',...
0572     'String','Get channels',...
0573     'FontSize', EditFontSize, ...
0574     'Callback', {@ltpdv_server_getChannels, mainfig}, ...
0575     'Units','normalized',...
0576     'Position',[bl bb bw bh]);
0577 end
0578 %-----
0579 % Build import from workspace panel
0580 function th = buildWorkspacePanel(hp, mainfig)
0581
0582 % Object list
0583 pmarg = 0.025;
0584 Lleft = pmarg;
0585 Lbottom = 0.4+pmarg;
0586 Lwidth = 1-2*pmarg;
0587 Lheight = 1-Lbottom-pmarg;
0588 th = uitable(hp, ...
0589     'Units','normalized',...
0590     'ColumnName',{'Name','Type','Size','kB'}, ...
0591     'Position',[Lleft Lbottom Lwidth Lheight],...
0592     'Tag','LTPDVworkspaceList',...
0593     'CellSelectionCallback', {@ltpdv_workspaceList_select, mainfig});
0594

```

```

0595 % set data
0596 ltpdv_workspaceList_set_data(th);
0597
0598 % Make refresh button
0599 pl = pmarg;
0600 ph = 0.05;
0601 pb = Lbottom - ph - pmarg;
0602 pw = 0.1;
0603 pbh = uicontrol(hp, 'Style', 'pushbutton', ...
0604 'String', 'Refresh', ...
0605 'Callback', {@ltpdv_workspaceList_set_data, th}, ...
0606 'Units', 'normalized', ...
0607 'Position', [pl pb pw ph]);
0608
0609 % Make import button
0610 pl = pmarg + pw + pmarg;
0611 ph = 0.05;
0612 pb = Lbottom - ph - pmarg;
0613 pw = 0.1;
0614 pbh = uicontrol(hp, 'Style', 'pushbutton', ...
0615 'String', 'Import', ...
0616 'Callback', {@ltpdv_importObjs, mainfig, th}, ...
0617 'Units', 'normalized', ...
0618 'Position', [pl pb pw ph]);
0619
0620 % Selection box for renamin with var names
0621 pl = pl + pw + pmarg;
0622 ph = 0.05;
0623 pb = Lbottom - ph - pmarg;
0624 pw = 0.3;
0625 ch = uicontrol(hp, 'Style', 'checkbox', ...
0626 'String', 'use var name', ...
0627 'TooltipString', 'Set object name to workspace variable name.', ...
0628 'Units', 'normalized', ...
0629 'Position', [pl pb pw ph], ...
0630 'Value', 1, ...
0631 'Tag', 'LTPDVusevarnameChk');
0632
0633 end
0634
0635
0636
0637
0638

```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_buildMathPanel.m

ltpdv_buildMathPanel

PURPOSE [^]

SYNOPSIS [^]

```
function ltpdv_buildMathPanel(panH, mainfig)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_calc_btn](#) LTPDV_CALC_BTN called when one of the calculator buttons is pushed.

This function is called by:

SOURCE CODE [^]

```

0001 function ltpdv_buildMathPanel(panH, mainfig)
0002
0003 Gproperties = getappdata(mainfig, 'Gproperties');
0004
0005 % Sizes
0006 pmarg = 0.025;           % Margin size
0007 pw = 0.5 - 2*pmarg;      % Panel width
0008 ph = 0.3;               % Panel height
0009 bheight = 0.2;          % Button height
0010 bwidth = 0.2;           % Button width
0011 BUTTON_FONT = 12;
0012 %----- Calculator pad
0013 pl = pmarg;
0014 pb = 0.5 + pmarg;
0015 % Panel
0016 pan = uipanel(panH, 'Title','Calculator ', 'FontSize', Gproperties.fontsize, ...
0017 'BackgroundColor', [0.4 0.4 0.4], ...
0018 'ForegroundColor', 'w', ...
0019 'Position',[pl pb pw ph], 'Units', 'normalized');
0020
0021 % Buttons
0022 buts = {'+', '-', './', '.*', '^.', '*', '/', '^'};
0023 bl = pmarg;
0024 bb = 1-pmarg-bheight;
0025 for j=1:numel(buts)
0026     bh = uicontrol(pan, 'String',buts{j}, ...
0027 'Callback', {@ltpdv_calc_btn, mainfig, 2}, ...
0028 'Units', 'normalized', 'FontSize', BUTTON_FONT, ...
0029 'Position',[bl bb bwidth bheight]);
0030     % move
0031     bl = bl + bwidth + pmarg;
0032     if bl + bwidth > (1-pmarg)
0033         bl = pmarg;
0034         bb = bb - bheight - pmarg;
0035     end
0036 end
0037
0038 %----- Stats panel
0039 pl = 0.5 + pmarg;
0040 pb = 0.5 + pmarg;
0041 % Panel
0042 pan = uipanel(panH, 'Title','Stats ', 'FontSize', Gproperties.fontsize, ...
0043 'BackgroundColor', [0.4 0.4 0.4], ...
0044 'ForegroundColor', 'w', ...
0045 'Position',[pl pb pw ph], 'Units', 'normalized');
0046
0047 % Buttons

```

```

0048 buts = {'mean', 'median', 'sum', 'std', 'var'};
0049 bl = pmarg;
0050 bb = 1-pmarg-bheight;
0051 for j=1:numel(buts)
0052     bh = uicontrol(pan, 'String',buts{j},...
0053         'Callback', {@ltpdv_calc_btn, mainfig, 1}, ...
0054         'Units', 'normalized', 'FontSize', BUTTON_FONT, ...
0055         'Position',[bl bb bwidth bheight]);
0056     % move
0057     bl = bl + bwidth + pmarg;
0058     if bl + bwidth > (1-pmarg)
0059         bl = pmarg;
0060         bb = bb - bheight - pmarg;
0061     end
0062 end
0063
0064 %----- Trig panel
0065 pl = pmarg;
0066 pb = pmarg;
0067 % Panel
0068 pan = uipanel(panH, 'Title','Trig ', 'FontSize', Gproperties.fontsize,...
0069     'BackgroundColor', [0.4 0.4 0.4],...
0070     'ForegroundColor', 'w',...
0071     'Position',[pl pb pw ph], 'Units', 'normalized');
0072
0073 % Buttons
0074 buts = {'sin', 'asin', 'cos', 'acos', 'tan', 'atan', 'log', 'log10', 'exp'};
0075 bl = pmarg;
0076 bb = 1-pmarg-bheight;
0077 for j=1:numel(buts)
0078     bh = uicontrol(pan, 'String',buts{j},...
0079         'Callback', {@ltpdv_calc_btn, mainfig, 1}, ...
0080         'Units', 'normalized', 'FontSize', BUTTON_FONT, ...
0081         'Position',[bl bb bwidth bheight]);
0082     % move
0083     bl = bl + bwidth + pmarg;
0084     if bl + bwidth > (1-pmarg)
0085         bl = pmarg;
0086         bb = bb - bheight - pmarg;
0087     end
0088 end
0089
0090 %----- Misc panel
0091 pl = 0.5 + pmarg;
0092 pb = pmarg;
0093 % Panel
0094 pan = uipanel(panH, 'Title','Misc ', 'FontSize', Gproperties.fontsize,...
0095     'BackgroundColor', [0.4 0.4 0.4],...
0096     'ForegroundColor', 'w',...
0097     'Position',[pl pb pw ph], 'Units', 'normalized');
0098
0099 % Buttons
0100 buts = {'sqrt', 'real', 'imag', 'conj', 'abs', 'phase'};
0101 bl = pmarg;
0102 bb = 1-pmarg-bheight;
0103 for j=1:numel(buts)
0104     bh = uicontrol(pan, 'String',buts{j},...
0105         'Callback', {@ltpdv_calc_btn, mainfig, 1}, ...
0106         'Units', 'normalized', 'FontSize', BUTTON_FONT, ...
0107         'Position',[bl bb bwidth bheight]);
0108     % move
0109     bl = bl + bwidth + pmarg;
0110     if bl + bwidth > (1-pmarg)
0111         bl = pmarg;
0112         bb = bb - bheight - pmarg;
0113     end
0114 end
0115

```

ltpdv_buildPlotPanel

PURPOSE ^

SYNOPSIS ^

```
function ltpdv_buildPlotPanel(panH, mainfig)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_buildPlotPanel(panH, mainfig)
0002
0003 Gproperties = getappdata(mainfig, 'Gproperties');
0004
0005 pmarg = 0.025;
0006
0007
0008
0009
```


ltpdv_buildPreprocessPanel

PURPOSE ^

SYNOPSIS ^

```
function ltpdv_buildPreprocessPanel(panH, mainfig)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_fcn_run](#) LTPDV_FCN_RUN runs the command on the selected panel
-  [ltpdv_preproc_fcnlist](#) Handles

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_buildPreprocessPanel(panH, mainfig)
0002
0003 Gproperties = getappdata(mainfig, 'Gproperties');
0004
0005 % List of functions
0006 preFcns = {'delay';...
0007            'downsample'; ...
0008            'dsmean'; ...
0009            'firwhiten'; ...
0010            'interp'; ...
0011            'join'; ...
0012            'polyfit'; ...
0013            'resample'; ...
0014            'select'; ...
0015            'smoother'; ...
0016            'split'; ...
0017            'timeshift'; ...
0018            'upsample'};
0019
0020 % Margins
0021 pmarg = 0.025;
0022 FONTSIZE = 12;
0023
0024 % Text string
0025 ll = pmarg;
0026 lh = 0.05;
0027 lb = 1-pmarg-lh;
0028 lw = 0.3;
0029 sth = uicontrol(panH, 'Style', 'text', ...
0030                'String', 'Functions', ...
0031                'Units', 'normalized', ...
0032                'BackgroundColor', Gproperties.Gcol, ...
0033                'FontSize', 14, ...
0034                'Position', [ll lb lw lh]);
0035
0036 % Params panel
0037 pl = pmarg + 0.3 + pmarg;
0038 pb = 0.1;
0039 pw = 1-pl-pmarg;
0040 ph = 1 - pb - pmarg;
0041
0042 paramh = uipanel('Parent', panH, 'Title', 'Parameters', ...
0043                 'Units', 'normalized', ...
0044                 'BackgroundColor', 0.8*[0.87 0.92 0.99], ...
0045                 'FontSize', FONTSIZE, ...
```

```

0046     'Tag', 'LTPDVpreprocparampan', ...
0047     'Position',[pl pb pw ph]);
0048
0049 % Function list
0050 ll = pmarg;
0051 lb = pmarg;
0052 lw = 0.3;
0053 lh = 1-2*pmarg - 0.05;
0054 lh = uicontrol(panH, 'Style', 'listbox', ...
0055     'String', preFcns, ...
0056     'BackgroundColor','w', ...
0057     'Value',1, 'Units', 'normalized', ...
0058     'FontSize', FONTSIZE, ...
0059     'Tag', 'LTPDVpreprocfcnlist', ...
0060     'Position',[ll lb lw lh], ...
0061     'Callback', {@ltpdv_preproc_fcnlist, paramh, mainfig});
0062
0063 % Build params
0064 ltpdv_preproc_fcnlist(lh, paramh, mainfig);
0065
0066 % Process button
0067 phght = 0.05;
0068 pbott = pmarg;
0069 pwidt = 0.2;
0070 pleft = 1-pmarg-pwidt;
0071 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0072     'String', 'Process', ...
0073     'Callback', {@ltpdv_fcn_run, 'preproc', panH, mainfig}, ...
0074     'Units', 'normalized', ...
0075     'Position',[pleft pbott pwidt phght]);
0076
0077
0078
0079 % END

```

ltpdv_buildPropertiesPanel

PURPOSE [^]

SYNOPSIS [^]

```
function ltpdv_buildPropertiesPanel(panH, mainfig)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SOURCE CODE [^]

```
0001 function ltpdv_buildPropertiesPanel(panH, mainfig)
0002
0003     EditFontSize = 11;
0004     vmarg      = 0.03;
0005     pmarg      = 0.01;
0006     CommHeight = EditFontSize/400;
0007     ph = 1-2*vmarg;
0008     htab = uipanel('Parent', panH, 'Position', [pmarg vmarg 1-2*pmarg ph]);
0009     set(htab, 'Units', 'Normalized')
0010     set(htab, 'Title', 'Properties for ')
0011     set(htab, 'BackgroundColor', [1 1 0.6]);
0012     set(htab, 'ForegroundColor', 'k');
0013     set(htab, 'Tag', 'LTPDVobjectProperties');
0014
0015
0016
0017
0018 end
0019
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_buildSpectralPanel.m

ltpdv_buildSpectralPanel

PURPOSE ^

SYNOPSIS ^

```
function ltpdv_buildSpectralPanel(panH, mainfig)
```

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_fcn_run](#) LTPDV_FCN_RUN runs the command on the selected panel
-  [ltpdv_spectral_fcncol](#) Handles

This function is called by:

SOURCE CODE ^

```

0001 function ltpdv_buildSpectralPanel(panH, mainfig)
0002
0003
0004 Gproperties = getappdata(mainfig, 'Gproperties');
0005
0006 % List of functions
0007 preFcns = {'psd'; ...
0008            'lpsd'; ...
0009            'tfe'; ...
0010            'ltfe'; ...
0011            'cohere'; ...
0012            'lcohere'; ...
0013            'cpsd'; ...
0014            'lcpsd'; ...
0015            'linedetect'; ...
0016            'rms'; ...
0017            };
0018
0019 % Margins
0020 pmarg = 0.025;
0021
0022 % Text string
0023 ll = pmarg;
0024 lh = 0.05;
0025 lb = 1-pmarg-lh;
0026 lw = 0.3;
0027 sth = uicontrol(panH, 'Style', 'text', ...
0028                'String', 'Functions', ...
0029                'Units', 'normalized', ...
0030                'BackgroundColor', Gproperties.Gcol, ...
0031                'FontSize', 14, ...
0032                'Position', [ll lb lw lh]);
0033
0034 % Params panel
0035 pl = pmarg + 0.3 + pmarg;
0036 pb = 0.1;
0037 pw = 1-pl-pmarg;
0038 ph = 1 - pb - pmarg;
0039
0040 paramh = uipanel('Parent', panH, 'Title', 'Parameters', ...
0041                'Units', 'normalized', ...
0042                'BackgroundColor', 0.8*[0.99 0.92 0.82], ...
0043                'FontSize', 14, ...
0044                'Tag', 'LTPDVspectralparampan', ...
0045                'Position', [pl pb pw ph]);

```

```

0046
0047 % Function list
0048 ll = pmarg;
0049 lb = pmarg;
0050 lw = 0.3;
0051 lh = 1-2*pmarg - 0.05;
0052 lh = uicontrol(panH, 'Style', 'listbox', ...
0053     'String', preFcns, ...
0054     'BackgroundColor', 'w', ...
0055     'Value', 1, 'Units', 'normalized', ...
0056     'FontSize', 14, ...
0057     'Tag', 'LTPDVspectralfcnlist', ...
0058     'Position', [ll lb lw lh], ...
0059     'Callback', {@ltpdv_spectral_fcnlist, paramh, mainfig});
0060
0061 % Build params
0062 ltpdv_spectral_fcnlist(lh, paramh, mainfig);
0063
0064 % Process button
0065 phght = 0.05;
0066 pbott = pmarg;
0067 pwidt = 0.2;
0068 pleft = 1-pmarg-pwidt;
0069 pbh = uicontrol(panH, 'Style', 'pushbutton', ...
0070     'String', 'Process', ...
0071     'Callback', {@ltpdv_fcn_run, 'spectral', panH, mainfig}, ...
0072     'Units', 'normalized', ...
0073     'Position', [pleft pbott pwidt phght]);
0074
0075
0076

```

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_build_buildobject.m

ltpdv_build_buildobject

PURPOSE [^]

LTPDV_BUILD_BUILD OBJECT callback for the build object button on the build

SYNOPSIS [^]

```
function ltpdv_build_buildobject(varargin)
```

DESCRIPTION [^]

LTPDV_BUILD_BUILD OBJECT callback for the build object button on the build panel.

M Hewitson 26-08-08

\$Id: ltpdv_build_buildobject.m,v 1.1 2008/09/02 07:51:47 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_build_get_selected_class](#) LTPDV_BUILD_GET_SELECTED_CLASS returns the currently selected class on
-  [ltpdv_get_params_from_panel](#) LTPDV_GET_PARAMS_FROM_PANEL get a plist from the parameters on a
-  [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE [^]

```
0001 % LTPDV_BUILD_BUILD OBJECT callback for the build object button on the build
0002 % panel.
0003 %
0004 % M Hewitson 26-08-08
0005 %
0006 % $Id: ltpdv_build_buildobject.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008
0009 function ltpdv_build_buildobject(varargin)
0010
0011     % Handles
0012     myh = varargin{1};
0013     mainfig = varargin{end};
0014     parampanh = findobj(mainfig, 'Tag', 'LTPDVbuildParamsPanel');
0015
0016     % Get class
0017     cl = ltpdv_build_get_selected_class(mainfig);
0018
0019     % Get parameters
0020     pl = ltpdv_get_params_from_panel(parampanh);
0021
0022     % Build this object
0023     obj = feval(cl, pl);
0024
0025     % Import object
0026     objs = getappdata(mainfig, 'LTPDAobjects');
0027     setappdata(mainfig, 'LTPDAobjects', [objs {obj}]);
0028     % Refresh object list
```

```
0029   ltpdv_refresh_object_list(mainfig);  
0030  
0031  
0032 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_build_class_select.m

ltpdv_build_class_select

PURPOSE [^]

LTPDV_BUILD_CLASS_SELECT callback when the user selects the class

SYNOPSIS [^]

```
function ltpdv_build_class_select(varargin)
```

DESCRIPTION [^]

```
LTPDV_BUILD_CLASS_SELECT callback when the user selects the class
of object to build

M Hewitson 26-08-08

$Id: ltpdv_build_class_select.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_build_get_selected_class](#) LTPDV_BUILD_GET_SELECTED_CLASS returns the currently selected class on
-  [ltpdv_build_set_select](#) LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE [^]

```
0001 % LTPDV_BUILD_CLASS_SELECT callback when the user selects the class
0002 % of object to build
0003 %
0004 % M Hewitson 26-08-08
0005 %
0006 % $Id: ltpdv_build_class_select.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008
0009 function ltpdv_build_class_select(varargin)
0010
0011 % Handles
0012 myh = varargin{1};
0013 mainfig = varargin{end};
0014
0015 % Fill in the set selector list
0016 sh = findobj(mainfig, 'Tag', 'LTPDVbuildSetSelect');
0017
0018 % Get the selected class
0019 cl = ltpdv_build_get_selected_class(mainfig);
0020
0021 % Get info object
0022 ii = eval([cl '.getInfo']);
0023
0024 % Set sets
0025 set(sh, 'Value', 1);
0026 set(sh, 'String', ii.sets);
0027
0028 % Callback
0029 ltpdv_build_set_select(sh, mainfig)
0030
```

0031 end

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_build_get_selected_class.m

ltpdv_build_get_selected_class

PURPOSE [^]

LTPDV_BUILD_GET_SELECTED_CLASS returns the currently selected class on

SYNOPSIS [^]

```
function cl = ltpdv_build_get_selected_class(mainfig)
```

DESCRIPTION [^]

LTPDV_BUILD_GET_SELECTED_CLASS returns the currently selected class on the build panel.

M Hewitson 26-08-08

\$Id: ltpdv_build_get_selected_class.m,v 1.1 2008/09/02 07:51:47 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

-  [ltpdv_build_buildobject](#) LTPDV_BUILD_BUILD_OBJECT callback for the build object button on the build
-  [ltpdv_build_class_select](#) LTPDV_BUILD_CLASS_SELECT callback when the user selects the class
-  [ltpdv_build_set_select](#) LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to

SOURCE CODE [^]

```
0001 % LTPDV_BUILD_GET_SELECTED_CLASS returns the currently selected class on
0002 % the build panel.
0003 %
0004 % M Hewitson 26-08-08
0005 %
0006 % $Id: ltpdv_build_get_selected_class.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008
0009 function cl = ltpdv_build_get_selected_class(mainfig)
0010
0011 % Fill in the set selector list
0012 sh = findobj(mainfig, 'Tag', 'LTPDVbuildObjectClass');
0013
0014 % Get the selected class
0015 str = get(sh, 'String');
0016 idx = get(sh, 'Value');
0017 cl = str{idx};
0018
0019 end
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_build_get_selected_pset.m

ltpdv_build_get_selected_pset

PURPOSE ^

LTPDV_BUILD_GET_SELECTED_PSET returns the currently selected set on

SYNOPSIS ^

```
function set = ltpdv_build_get_selected_pset(mainfig)
```

DESCRIPTION ^

```
LTPDV_BUILD_GET_SELECTED_PSET returns the currently selected set on
the build panel.

M Hewitson 26-08-08

$Id: ltpdv_build_get_selected_pset.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_build_set_select](#) LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to

SOURCE CODE ^

```
0001 % LTPDV_BUILD_GET_SELECTED_PSET returns the currently selected set on
0002 % the build panel.
0003 %
0004 % M Hewitson 26-08-08
0005 %
0006 % $Id: ltpdv_build_get_selected_pset.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008
0009 function set = ltpdv_build_get_selected_pset(mainfig)
0010
0011 % Fill in the set selector list
0012 sh = findobj(mainfig, 'Tag', 'LTPDVbuildSetSelect');
0013
0014 % Get the selected class
0015 strs = get(sh, 'String');
0016 idx = get(sh, 'Value');
0017 set = strs{idx};
0018
0019 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_build_params_panel.m

ltpdv_build_params_panel

PURPOSE [^]

SYNOPSIS [^]

```
function ltpdv_build_params_panel(ph, ii, varargin)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_preproc_param_act](#) Handles
-  [ltpdv_preproc_sets_callback](#)

This function is called by:

-  [ltpdv_build_set_select](#) LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to
-  [ltpdv_preproc_fcncall](#) Handles
-  [ltpdv_preproc_sets_callback](#)
-  [ltpdv_spectral_fcncall](#) Handles

SOURCE CODE [^]

```
0001 function ltpdv_build_params_panel(ph, ii, varargin)
0002
0003     FONTSIZE = 12;
0004
0005     pmarg = 0.025;
0006
0007     keyl = pmarg;
0008     keyh = 0.055;
0009     keyw = 0.3;
0010     keyb = 1-pmarg-keyh;
0011
0012     vall = pmarg*2 + keyw;
0013     valw = 0.5;
0014
0015     al    = vall + valw + pmarg;
0016     aw    = 0.05;
0017
0018     delete(get(ph, 'Children'))
0019
0020     if numel(varargin) > 0
0021         activated = varargin{1};
0022         if strcmpi(activated, 'on')
0023             chkVal = 1;
0024         else
0025             chkVal = 0;
0026         end
0027     else
0028         activated = 'off';
0029         chkVal = 0;
0030     end
0031
0032     pancomps = {};
0033
0034     if isa(ii, 'minfo')
0035         % we give a pop-up menu of sets
```

```

00036     eh = uicontrol(ph, 'Style', 'popupmenu', ...
00037     'String', ii.sets, ...
00038     'units', 'normalized', ...
00039     'BackgroundColor', 'w', ...
00040     'FontSize', FONTSIZE, ...
00041     'Enable', 'on', ...
00042     'Position', [vall keyb valw keyh]);
00043
00044     set(eh, 'Callback', {@ltpdv_preproc_sets_callback, ii});
00045 else
00046     pl = ii;
00047
00048     if numel(pl.params) == 0
00049         sth = uicontrol(ph, 'Style', 'text', ...
00050         'String', 'contains no parameters', ...
00051         'Units', 'normalized', ...
00052         'BackgroundColor', 'w', ...
00053         'FontSize', FONTSIZE, ...
00054         'Position', [keyl keyb 2*keyw keyh]);
00055     end
00056     % check for any unsupported parameters
00057     unsupported = false;
00058     for j=1:numel(pl.params)
00059         val = pl.params(j).val;
00060         if isa(val, 'timespan')
00061             unsupported = true;
00062         end
00063     end
00064     if unsupported
00065         sth = uicontrol(ph, 'Style', 'text', ...
00066         'String', ['contains unsupported parameter of type: ' class(val)], ...
00067         'Units', 'normalized', ...
00068         'BackgroundColor', 'w', ...
00069         'FontSize', FONTSIZE, ...
00070         'Position', [keyl keyb 2*keyw keyh]);
00071     else
00072         % Build parameters
00073         for j=1:numel(pl.params)
00074
00075             % Get key and val
00076             key = pl.params(j).key;
00077             val = pl.params(j).val;
00078
00079             switch class(val)
00080             case 'char'
00081                 valstr = val;
00082             case 'double'
00083                 valstr = mat2str(val);
00084             case 'specwin'
00085                 valstr = specwin.getTypes;
00086             otherwise
00087                 valstr = char(val);
00088             end
00089
00090             % key text
00091             sth = uicontrol(ph, 'Style', 'text', ...
00092             'String', key, ...
00093             'Units', 'normalized', ...
00094             'BackgroundColor', 'w', ...
00095             'FontSize', FONTSIZE, ...
00096             'Position', [keyl keyb keyw keyh]);
00097
00098             % val edit
00099             if ischar(valstr) || isa(valstr, 'time') || isa(valstr, 'sym')
00100                 if isa(valstr, 'time') || isa(valstr, 'sym')
00101                     valstr = char(valstr);
00102                 end
00103                 eh = uicontrol(ph, 'Style', 'edit', ...
00104                 'String', valstr, ...
00105                 'units', 'normalized', ...
00106                 'BackgroundColor', 'w', ...
00107                 'FontSize', FONTSIZE, ...
00108                 'Enable', 'activated', ...
00109                 'Position', [vall keyb valw keyh]);
00110             elseif iscell(valstr)
00111                 eh = uicontrol(ph, 'Style', 'popupmenu', ...
00112                 'String', valstr, ...
00113                 'units', 'normalized', ...
00114                 'BackgroundColor', 'w', ...
00115                 'FontSize', FONTSIZE, ...
00116                 'Enable', 'activated', ...
00117                 'Position', [vall keyb valw keyh]);
00118             else
00119                 valstr
00120                 error('### Unknown type for value string.');
```

```
0122     end
0123     setappdata(eh, 'valClass', class(val));
0124     % Activate
0125     ah = uicontrol(ph, 'Style', 'checkbox', ...
0126         'String', '', ...
0127         'units', 'normalized' ...
0128         'BackgroundColor', get(ph, 'BackgroundColor'), ...
0129         'FontSize', FONTSIZE, ...
0130         'Value', chkVal, 'Position', [al keyb aw keyh], ...
0131         'Callback', {@ltpdv_preproc_param_act, ph});
0132
0133     pancomps = [pancomps; {sth, eh, ah}];
0134
0135     % Do next line
0136     keyb = keyb - pmarg - keyh;
0137
0138     end
0139 end
0140 end
0141
0142     setappdata(ph, 'pancomps', pancomps);
0143
0144
0145 end
```

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_build_set_select.m

ltpdv_build_set_select

PURPOSE [^]

LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to

SYNOPSIS [^]

function ltpdv_build_set_select(varargin)

DESCRIPTION [^]

LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to work with.

M Hewitson 26-08-08

\$Id: ltpdv_build_set_select.m,v 1.1 2008/09/02 07:51:47 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_build_get_selected_class](#) LTPDV_BUILD_GET_SELECTED_CLASS returns the currently selected class on
-  [ltpdv_build_get_selected_pset](#) LTPDV_BUILD_GET_SELECTED_PSET returns the currently selected set on
-  [ltpdv_build_params_panel](#)

This function is called by:

-  [ltpdv_buildImportPanel](#)
-  [ltpdv_build_class_select](#) LTPDV_BUILD_CLASS_SELECT callback when the user selects the class

SOURCE CODE [^]

```
0001 % LTPDV_BUILD_SET_SELECT callback when the user selects a parameter set to
0002 % work with.
0003 %
0004 % M Hewitson 26-08-08
0005 %
0006 % $Id: ltpdv_build_set_select.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008 function ltpdv_build_set_select(varargin)
0009
0010 % Handles
0011 myh = varargin{1};
0012 mainfig = varargin{end};
0013
0014 % Get selected class
0015 cl = ltpdv_build_get_selected_class(mainfig);
0016
0017 % Get selected set
0018 set = ltpdv_build_get_selected_pset(mainfig);
0019
0020 % Get parameter set
0021 ii = eval([cl '.getInfo('' cl '', '' set '')']);
0022 pl = ii.plists;
0023
0024 % Build parameter list
0025 ph = findobj(mainfig, 'Tag', 'LTPDVbuildParamsPanel');
0026 ltpdv_build_params_panel(ph, pl, 'on')
0027
```

0028 [end](#)

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_calc_btn.m

ltpdv_calc_btn

PURPOSE ^

LTPDV_CALC_BTN called when one of the calculator buttons is pushed.

SYNOPSIS ^

```
function ltpdv_calc_btn(varargin)
```

DESCRIPTION ^

```
LTPDV_CALC_BTN called when one of the calculator buttons is pushed.
M Hewitson 22-04-08
$Id: ltpdv_calc_btn.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.
-  [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

-  [ltpdv_buildMathPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_calc_btn(varargin)
0002 % LTPDV_CALC_BTN called when one of the calculator buttons is pushed.
0003 %
0004 % M Hewitson 22-04-08
0005 %
0006 % $Id: ltpdv_calc_btn.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0007 %
0008 %
0009 % Handles
0010 myh      = varargin{1};
0011 mainfig = varargin{end-1};
0012 nins     = varargin{end};
0013 %
0014 % Get operation
0015 op = get(myh, 'String');
0016 %
0017 % Get selected objects
0018 objs = ltpdv_get_selected_objs(mainfig);
0019 Nobjs = length(objs);
0020 %
0021 if Nobjs < nins
0022     error('### Please select two objects');
0023 end
0024 %
0025 % check objects are all AOs
0026 for j=1:Nobjs
0027     if ~isa(objs{j}, 'ao')
0028         error('### Please select only AOs for math calculations');
0029     end
0030 end
0031 %
0032 % Loop over AOs
0033 n = 1;
```



```
0034 res = {};
0035 while n <= length(objs)
0036     % operate on two objects
0037     switch nins
0038     case 1
0039         res = [res {feval(op, objs{n})}];
0040     case 2
0041         if isempty(res)
0042             res = feval(op, objs{n}, objs{n+1});
0043             n = n + 1;
0044         else
0045             res = feval(op, res, objs{n});
0046         end
0047     end
0048     n = n + 1;
0049 end
0050
0051 if nins == 2
0052     res = {res};
0053 end
0054
0055 % Add this result to the object list
0056 objs = getappdata(mainfig, 'LTPDAobjects');
0057 if ~isempty(res)
0058     setappdata(mainfig, 'LTPDAobjects', [objs res]);
0059     % Refresh object list
0060     ltpdv_refresh_object_list(mainfig);
0061 end
0062
0063
0064 % END
```

ltpdv_cb_mainfigClose

PURPOSE ^

Callback executed when the main figure closes

SYNOPSIS ^

function ltpdv_cb_mainfigClose(varargin)

DESCRIPTION ^

Callback executed when the main figure closes
M Hewitson
\$Id: ltpdv_cb_mainfigClose.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_cb_mainfigClose(varargin)
0002
0003 % Callback executed when the main figure closes
0004 %
0005 % M Hewitson
0006 %
0007 % $Id: ltpdv_cb_mainfigClose.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 disp('* Goodbye from the LTPDA Dataviewer *')
0011 delete(varargin{1})
0012
0013
0014
0015
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_delete_objects.m

ltpdv_delete_objects

PURPOSE ^

LTPDV_DELETE_OBJECTS delete the selected objects from the object list

SYNOPSIS ^

function ltpdv_delete_objects(varargin)

DESCRIPTION ^

```
LTPDV_DELETE_OBJECTS delete the selected objects from the object list
M Hewitson 22-04-08
$Id: ltpdv_delete_objects.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_delete_objects(varargin)
0002
0003 % LTPDV_DELETE_OBJECTS delete the selected objects from the object list
0004 %
0005 % M Hewitson 22-04-08
0006 %
0007 % $Id: ltpdv_delete_objects.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 % Handles
0011 clearAll = false;
0012 if nargin == 3
0013     myh = varargin{1};
0014     mainfig = varargin{end};
0015 elseif nargin == 4
0016     myh = varargin{1};
0017     mainfig = varargin{end-1};
0018     if strcmp(varargin{end}, 'all')
0019         clearAll = true;
0020     end
0021 end
0022
0023 % Get selected rows
0024 rows = getappdata(mainfig, 'ObjectListSelectedRows');
0025
0026 if clearAll
0027     % Set new object list
0028     setappdata(mainfig, 'LTPDAobjects', {});
0029 else
0030     % Get current object list
0031     objs = getappdata(mainfig, 'LTPDAobjects');
0032
0033     % Build index list to keep
0034     a = 1:length(objs);
0035     idx = a(~ismember(a,rows));
0036     % Set new object list
0037     setappdata(mainfig, 'LTPDAobjects', objs(idx));
0038 end
0039
```

```
0040  
0041 % Refresh object list  
0042 ltpdv_refresh_object_list(mainfig);  
0043  
0044 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_explore_objects.m

ltpdv_explore_objects

PURPOSE ^

Handles

SYNOPSIS ^

```
function ltpdv_explore_objects(varargin)
```

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:



[ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_explore_objects(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007
0008 % Get selected objects
0009 objs = ltpdv_get_selected_objs(mainfig);
0010
0011 % Pass to ltpda_explorer
0012 ltpda_explorer(objs)
0013
0014 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_export_objects

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_export_objects(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_export_objects(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007
0008 % check current workspace variables
0009 basename = 'ex_obj';
0010 ex = 0;
0011 n = [];
0012 vars = evalin('base', 'who');
0013 for j=1:length(vars)
0014     v = char(vars{j});
0015     if strcmp(basename, v, length(basename)) == 1
0016         % get the number
0017         n = [n str2num(v(end-1:end))];
0018     end
0019 end
0020
0021 % Get selected objects
0022 objs = ltpdv_get_selected_objs(mainfig);
0023
0024 % Loop over selected objects
0025 for j=1:numel(objs)
0026     obj = objs{j};
0027     ex = max([n ex])+1;
0028     varname = sprintf('%s_%02d', basename, ex);
0029     assignin('base', varname, obj);
0030 end
0031
0032
0033 % END
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_fcn_run.m

ltpdv_fcn_run

PURPOSE [^]

LTPDV_FCN_RUN runs the command on the selected panel

SYNOPSIS [^]

```
function ltpdv_fcn_run(varargin)
```

DESCRIPTION [^]

```
LTPDV_FCN_RUN runs the command on the selected panel
M Hewitson 23-04-08
$Id: ltpdv_fcn_run.m,v 1.3 2008/09/02 07:51:47 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_get_params_from_panel](#) LTPDV_GET_PARAMS_FROM_PANEL get a plist from the parameters on a
-  [ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.
-  [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

-  [ltpdv_buildPreprocessPanel](#)
-  [ltpdv_buildSpectralPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_fcn_run(varargin)
0002 % LTPDV_FCN_RUN runs the command on the selected panel
0003 %
0004 % M Hewitson 23-04-08
0005 %
0006 % $Id: ltpdv_fcn_run.m,v 1.3 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008
0009 % Get handles
0010 myh = varargin{1};
0011 tag = varargin{end-2};
0012 panH = varargin{end-1};
0013 mainfig = varargin{end};
0014
0015 % Get function
0016 switch tag
0017 case 'preproc'
0018     listtag = 'LTPDVpreprocfcnlist';
0019     ptag = 'LTPDVpreprocparampan';
0020 case 'spectral'
0021     listtag = 'LTPDVspectralfcnlist';
0022     ptag = 'LTPDVspectralparampan';
0023 otherwise
0024     error('### Unknown function set');
0025 end
0026 fcnh = findobj(panH, 'Tag', listtag);
0027 val = get(fcnh, 'Value');
0028 strs = get(fcnh, 'String');
```

```
0029 fcn = str{val};
0030
0031 % Get params
0032 parampanh = findobj(panH, 'Tag', ptag);
0033 pl = ltpdv_get_params_from_panel(parampanh);
0034
0035 % Get selected objects
0036 objs = ltpdv_get_selected_objs(mainfig);
0037
0038 % Collect AOs
0039 as = [];
0040 for j=1:numel(objs)
0041     obj = objs{j};
0042     if isa(obj, 'ao')
0043         as = [as obj];
0044     else
0045         warning('!!! Skipping object %s - it's not an AO', obj.name);
0046     end
0047 end
0048
0049 % Evaluate fcn
0050 res = feval(fcn, as, pl);
0051
0052 % Add result to object list
0053 for j=1:numel(res)
0054     objs = getappdata(mainfig, 'LTPDAobjects');
0055     setappdata(mainfig, 'LTPDAobjects', [objs {res(j)}]);
0056 end
0057 % Refresh object list
0058 ltpdv_refresh_object_list(mainfig);
0059
0060 end
0061
0062 % END
```

ltpdv_filterAdd

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_filterAdd(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_get_selected_filter](#) Get selected panel
-  [ltpdv_set_filter_list](#) Get filters

This function is called by:

-  [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_filterAdd(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig = varargin{end};
0007
0008 % Get existing filters
0009 filts = getappdata(mainfig, 'Filters');
0010
0011 % Get current filter
0012 filt = ltpdv_get_selected_filter(htab);
0013
0014 % Check name
0015 count = 1;
0016 while true
0017     found = false;
0018     for j=1:length(filts)
0019         if strcmp(filt.name, filts(j).name)
0020             if count > 1
0021                 filt.setName(sprintf('%s%02d', filt.name(1:end-2), count), 'internal');
0022             else
0023                 filt.setName(sprintf('%s%02d', filt.name, count), 'internal');
0024             end
0025             found = true;
0026             count = count + 1;
0027         end
0028     end
0029     if ~found
0030         break;
0031     end
0032 end
0033
0034 % Add to list
0035 setappdata(mainfig, 'Filters', [filts filt]);
0036
0037 % Refresh list
0038 ltpdv_set_filter_list(mainfig);
```



Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_filterApply

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_filterApply(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.
-  [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

-  [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_filterApply(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig = varargin{end};
0007
0008 % Get selected filter
0009 filts = getappdata(mainfig, 'Filters');
0010 % Get selection
0011 fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
0012 idx = get(fh, 'Value');
0013 sfilts = [];
0014 for j=1:length(filts)
0015     if ismember(j, idx)
0016         sfilts = [sfilts filts(j)];
0017     end
0018 end
0019
0020 %----- Apply filter to selected objects
0021
0022 % Get selected objects
0023 objs = ltpdv_get_selected_objs(mainfig);
0024
0025 % Collect AOs
0026 as = [];
0027 for j=1:numel(objs)
0028     obj = objs{j};
0029     if isa(obj, 'ao')
0030         as = [as obj];
0031     else
0032         warning('!!! Skipping object %s - it''s not an AO', obj.name);
0033     end
0034 end
0035
0036 % Evaluate fcn
0037 out = as;
```

```
0038 for j=1:length(sfilts)
0039     filt = sfilts(j);
0040     out = feval('filter', out, plist('filter', filt));
0041 end
0042
0043 % Add result to object list
0044 for j=1:numel(out)
0045     objs = getappdata(mainfig, 'LTPDAobjects');
0046     setappdata(mainfig, 'LTPDAobjects', [objs {out(j)}]);
0047 end
0048 % Refresh object list
0049 ltpdv_refresh_object_list(mainfig);
0050
0051
0052
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_filterBrowse

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_filterBrowse(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_filterBrowse(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 panh     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % File browser
0009 [filename, pathname, filterindex] = uigetfile( ...
0010     {'*.fil', 'LISO Files'; '*.xml', 'LTPDA XML File'; '*.mat', 'MAT-File'}, 'Select
Filter File');
0011
0012 if isequal(filename,0) || isequal(pathname,0)
0013 else
0014     % Full file path
0015     fpath = fullfile(pathname, filename);
0016
0017     % Set file name in file box
0018     fbh = findobj(panh, 'Tag', 'LTPDVfilterFileFilename');
0019     set(fbh, 'String', filename);
0020
0021     % Load filter
0022     filt = miir(fpath);
0023     filt.name = filename;
0024
0025     % Store filter
0026     setappdata(panh, 'filter', filt);
0027
0028     % Filter info
0029     fih = findobj(panh, 'Tag', 'LTPDVfilterFileInfo');
0030     set(fih, 'String', display(filt));
0031 end
0032
0033
0034
0035
0036
0037
```


ltpdv_filterDelete

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_filterDelete(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_filter_list](#) Get filters

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_filterDelete(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get selection
0009 fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
0010 idx = get(fh, 'Value');
0011
0012 % Get filters
0013 filts = getappdata(mainfig, 'Filters');
0014
0015 fo = [];
0016 for j=1:length(filts)
0017     if j~=idx
0018         fo = [fo filts(j)];
0019     end
0020 end
0021
0022 setappdata(mainfig, 'Filters', fo);
0023 set(fh, 'Value', idx-1);
0024 % Refresh list
0025 ltpdv_set_filter_list(mainfig);
0026
```


ltpdv_filterImport

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_filterImport(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_filterImport(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 panh     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get selected objects
0009 filts = ltpdv_get_selected_objs(mainfig);
0010 if length(filts) > 1
0011     error('### Select a single filter object');
0012 end
0013 filt = filts{1};
0014 if ~isa(filt, 'miir')
0015     error('### Select a single miir filter object');
0016 end
0017
0018 % Store filter
0019 setappdata(panh, 'filter', filt);
0020
0021 % Filter info
0022 fih = findobj(panh, 'Tag', 'LTPDVfilterObjectInfo');
0023 set(fih, 'String', display(filt));
0024
```


ltpdv_filter_control_chans

PURPOSE ^

DV_FILTERCONTROLCHANS filter out control channels

SYNOPSIS ^

function chansout = filter_control_chans(channels)

DESCRIPTION ^

DV_FILTERCONTROLCHANS filter out control channels
M Hewitson 26-07-06
\$Id: ltpdv_filter_control_chans.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_server_set_channel_list](#) Get normal channels

SOURCE CODE ^

```
0001 function chansout = filter_control_chans(channels)
0002
0003 % DV_FILTERCONTROLCHANS filter out control channels
0004 %
0005 % M Hewitson 26-07-06
0006 %
0007 % $Id: ltpdv_filter_control_chans.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 nchans = size(channels,1);
0011 chansout = [];
0012 for c=1:nchans
0013     idx = strfind(channels(c,:), '#');
0014     if isempty(idx)
0015         chansout = strvcats(chansout, deblank(channels(c,:)));
0016     end
0017 end
0018
0019 chansout = char(sort(cellstr(chansout)));
0020
0021 % END
```


ltpdv_filter_from_standard_panel

PURPOSE ^

LTPDV_FILTER_FROM_STANDARD_PANEL builds a filter from the description on

SYNOPSIS ^

function filt = ltpdv_filter_from_standard_panel(panh)

DESCRIPTION ^

LTPDV_FILTER_FROM_STANDARD_PANEL builds a filter from the description on the standard filter panel.

M Hewitson 24-04-08

\$Id: ltpdv_filter_from_standard_panel.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_get_selected_filter](#) Get selected panel

SOURCE CODE ^

```
0001 function filt = ltpdv_filter_from_standard_panel(panh)
0002 % LTPDV_FILTER_FROM_STANDARD_PANEL builds a filter from the description on
0003 % the standard filter panel.
0004 %
0005 % M Hewitson 24-04-08
0006 %
0007 % $Id: ltpdv_filter_from_standard_panel.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010
0011 % Filter type
0012 h = findobj(panh, 'Tag', 'LTPDVfiltStandardType');
0013 idx = get(h, 'Value');
0014 strs = get(h, 'String');
0015 ftype = strs{idx};
0016
0017 disp(sprintf('** Building %s filter', ftype))
0018
0019 % Gain
0020 h = findobj(panh, 'Tag', 'LTPDVfiltStandardGain');
0021 gain = str2double(get(h, 'String'));
0022
0023 % Fc
0024 h = findobj(panh, 'Tag', 'LTPDVfiltStandardCutoff');
0025 fc = str2num(get(h, 'String'));
0026
0027 % Fs
0028 h = findobj(panh, 'Tag', 'LTPDVfiltStandardFs');
0029 fs = str2double(get(h, 'String'));
0030
0031 % Order
0032 h = findobj(panh, 'Tag', 'LTPDVfiltStandardOrder');
0033 order = str2double(get(h, 'String'));
0034
0035 % Ripple
0036 h = findobj(panh, 'Tag', 'LTPDVfiltStandardRipple');
0037 ripple = str2double(get(h, 'String'));
0038
```



```
0039 % Build plist
0040 pl = plist('type', lower(ftype), ...
0041           'Gain', gain, ...
0042           'fc', fc, ...
0043           'fs', fs, ...
0044           'order', order, ...
0045           'ripple', ripple);
0046
0047 % Build filter
0048 filt = miir(pl);
0049
0050 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_filter_get_pzmodel

PURPOSE ^

Get poles

SYNOPSIS ^

function pzm = ltpdv_filter_get_pzmodel(mainfig)

DESCRIPTION ^

Get poles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_get_selected_filter](#) Get selected panel
-  [ltpdv_pzmResp](#) Handles
-  [ltpdv_pzmStore](#) Handles
-  [ltpdv_pzmStoreResp](#) Handles

SOURCE CODE ^

```
0001 function pzm = ltpdv_filter_get_pzmodel(mainfig)
0002
0003 % Get poles
0004 poles = getappdata(mainfig, 'FilterPoles');
0005
0006 % Get zeros
0007 zeros = getappdata(mainfig, 'FilterZeros');
0008
0009 % Get gain
0010 gh = findobj(mainfig, 'Tag', 'LTPDVfilterPZgain');
0011 gstr = get(gh, 'String');
0012 if ~isempty(gstr)
0013     gain = num2str(gstr);
0014 else
0015     gain = 1;
0016 end
0017
0018 % Make model
0019 pzm = pzmodel(gain, poles, zeros);
0020
0021 % Get name
0022 nh = findobj(mainfig, 'Tag', 'LTPDVfilterPZname');
0023 nstr = get(nh, 'String');
0024 if ~isempty(nstr)
0025     pzm.name = nstr;
0026 end
0027
0028
0029
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_filter_getf.m

ltpdv_filter_getf

PURPOSE ^

get f1

SYNOPSIS ^

```
function pl = ltpdv_filter_getf(panh)
```

DESCRIPTION ^

```
get f1
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_plotFilterResp](#) LTPDV_PLOTFILTERRESP plots the response of the selected filter.
-  [ltpdv_storeFilterResp](#) LTPDV_STOREFILTERRESP stores the response on the current filter in the

SOURCE CODE ^

```
0001 function pl = ltpdv_filter_getf(panh)
0002
0003 % get f1
0004 h = findobj(panh, 'Tag', 'LTPDVfilterF1');
0005 f1 = str2num(get(h, 'String'));
0006 % get f2
0007 h = findobj(panh, 'Tag', 'LTPDVfilterF2');
0008 f2 = str2num(get(h, 'String'));
0009 % get nf
0010 h = findobj(panh, 'Tag', 'LTPDVfilterNf');
0011 nf = str2num(get(h, 'String'));
0012 % get scale
0013 h = findobj(panh, 'Tag', 'LTPDVfilterScale');
0014 idx = get(h, 'Value');
0015 strs = get(h, 'String');
0016 scale = strs{idx};
0017
0018 pl = plist('scale', scale);
0019 if ~isempty(f1)
0020     pl = append(pl, 'f1', f1);
0021 end
0022 if ~isempty(f2)
0023     pl = append(pl, 'f2', f2);
0024 end
0025 if ~isempty(nf)
0026     pl = append(pl, 'nf', nf);
0027 end
0028
0029 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_filter_list_select

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_filter_list_select(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_filter_list_select(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get filters
0008 filts = getappdata(mainfig, 'Filters');
0009
0010 % Get selected index
0011 idx = get(myh, 'Value');
0012
0013 filts(idx)
0014
```


ltpdv_filtersClear

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_filtersClear(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_filter_list](#) Get filters

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_filtersClear(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Clear list
0009 setappdata(mainfig, 'Filters', []);
0010
0011 % Refresh list
0012 ltpdv_set_filter_list(mainfig);
0013
0014
```


ltpdv_getWorkspaceObjs

PURPOSE ^

LTPDV_GETWORKSPACEOBSJS gets a list of LTPDA user objects currently in the

SYNOPSIS ^

function objs = ltpdv_getWorkspaceObjs()

DESCRIPTION ^

LTPDV_GETWORKSPACEOBSJS gets a list of LTPDA user objects currently in the MATLAB workspace.

M Hewitson 22-04-08

\$Id: ltpdv_getWorkspaceObjs.m,v 1.2 2008/06/18 15:07:35 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_workspaceList_set_data](#) LTPDV_WORKSPACELIST_SET_DATA sets the data into the workspace table

SOURCE CODE ^

```
0001 function objs = ltpdv_getWorkspaceObjs()
0002 % LTPDV_GETWORKSPACEOBSJS gets a list of LTPDA user objects currently in the
0003 % MATLAB workspace.
0004 %
0005 % M Hewitson 22-04-08
0006 %
0007 % $Id: ltpdv_getWorkspaceObjs.m,v 1.2 2008/06/18 15:07:35 hewitson Exp $
0008 %
0009
0010 % get base workspace variables
0011 ws_vars = evalin('base','whos');
0012
0013 objs = [];
0014 for j=1:length(ws_vars)
0015
0016     cmd = sprintf('obj = evalin(''base'', '%s');', ws_vars(j).name);
0017     eval(cmd)
0018     if isa(obj, 'ltpda_uo')
0019         objs = [objs ws_vars(j)];
0020     end
0021
0022 end
0023
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_get_params_from_panel.m

ltpdv_get_params_from_panel

PURPOSE [^]

LTPDV_GET_PARAMS_FROM_PANEL get a plist from the parameters on a

SYNOPSIS [^]

```
function pl = ltpdv_get_params_from_panel(parampanh)
```

DESCRIPTION [^]

LTPDV_GET_PARAMS_FROM_PANEL get a plist from the parameters on a
parameters panel

M Hewitson 26-08-08

\$Id: ltpdv_get_params_from_panel.m,v 1.1 2008/09/02 07:51:47 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

-  [ltpdv_build_buildobject](#) LTPDV_BUILD_BUILD OBJECT callback for the build object button on the build
-  [ltpdv_fcn_run](#) LTPDV_FCN_RUN runs the command on the selected panel

SOURCE CODE [^]

```
0001 % LTPDV_GET_PARAMS_FROM_PANEL get a plist from the parameters on a
0002 % parameters panel
0003 %
0004 % M Hewitson 26-08-08
0005 %
0006 % $Id: ltpdv_get_params_from_panel.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008 function pl = ltpdv_get_params_from_panel(parampanh)
0009
0010
0011 % Get params
0012 pancomps = getappdata(parampanh, 'pancomps');
0013
0014 % loop over params
0015 pl = plist();
0016 for j=1:size(pancomps,1)
0017
0018 % handles
0019 keyf = pancomps{j,1};
0020 valf = pancomps{j,2};
0021 actf = pancomps{j,3};
0022
0023 % Get activate status
0024 if get(actf, 'Value')
0025 % Get key name
0026 key = get(keyf, 'String');
0027
0028 % Get param value
0029 valclass = getappdata(valf, 'valClass');
0030
0031 switch valclass
0032 case 'char'
0033     val = get(valf, 'String');
0034 case 'double'
0035     val = get(valf, 'String');
```

```
0036         try
0037             val = eval(val);
0038         end
0039         case 'sym'
0040             val = get(valf, 'String');
0041             val = sym(val);
0042         case 'specwin'
0043             str = get(valf, 'String');
0044             idx = get(valf, 'Value');
0045             val = str{idx};
0046             if strcmp(val, 'Kaiser')
0047                 val = specwin(val, 10, 150);
0048             else
0049                 val = specwin(val, 10);
0050             end
0051         case 'time'
0052             val = get(valf, 'String');
0053             val = time(val);
0054         otherwise
0055             error(['### Unsupported value class: ' valclass]);
0056         end
0057     % Add to plist
0058     pl = append(pl, key, val);
0059 end
0060 end
0061 end
0062 end
0063 end
```

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_get_selected_filter.m

ltpdv_get_selected_filter

PURPOSE [^]

Get selected panel

SYNOPSIS [^]

```
function filt = ltpdv_get_selected_filter(htab)
```

DESCRIPTION [^]

Get selected panel

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_filter_from_standard_panel](#) LTPDV_FILTER_FROM_STANDARD_PANEL builds a filter from the description on
-  [ltpdv_filter_get_pzmodel](#) Get poles

This function is called by:

-  [ltpdv_filterAdd](#) Handles

SOURCE CODE [^]

```
0001 function filt = ltpdv_get_selected_filter(htab)
0002
0003
0004 % Get selected panel
0005 pidx = get(htab, 'SelectedIndex');
0006 pans = get(htab, 'Children');
0007 panh = pans(pidx);
0008
0009
0010 % Get filter from panel handle
0011 switch get(panh, 'Title')
0012     case 'Standard'
0013         % Build filter from standard panel
0014         filt = ltpdv_filter_from_standard_panel(panh);
0015     case 'File'
0016         filt = getappdata(panh, 'filter');
0017     case 'Object'
0018         filt = getappdata(panh, 'filter');
0019     case 'Poles/Zeros'
0020         mainfig = findobj(0, 'Tag', 'LTPDVmainfig');
0021         pzm = ltpdv_filter_get_pzmodel(mainfig);
0022         filt = miir(plist('pzmodel', pzm));
0023     otherwise
0024         error('### Unknown panel.');
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_get_selected_objs.m

ltpdv_get_selected_objs

PURPOSE ^

LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.

SYNOPSIS ^

```
function objs = ltpdv_get_selected_objs(mainfig)
```

DESCRIPTION ^

```
LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.
M Hewitson 22-04-08
$Id: ltpdv_get_selected_objs.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_calc_btn](#) LTPDV_CALC_BTN called when one of the calculator buttons is pushed.
-  [ltpdv_explore_objects](#) Handles
-  [ltpdv_export_objects](#) Handles
-  [ltpdv_fcn_run](#) LTPDV_FCN_RUN runs the command on the selected panel
-  [ltpdv_filterApply](#) Handles
-  [ltpdv_filterImport](#) Handles
-  [ltpdv_hist_plot](#) Handles
-  [ltpdv_plot](#) Handles
-  [ltpdv_save_objects](#) Handles

SOURCE CODE ^

```
0001 function objs = ltpdv_get_selected_objs(mainfig)
0002
0003 % LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.
0004 %
0005 % M Hewitson 22-04-08
0006 %
0007 % $Id: ltpdv_get_selected_objs.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 % Get selected rows
0011 rows = getappdata(mainfig, 'ObjectListSelectedRows');
0012 % Get current object list
0013 objs = getappdata(mainfig, 'LTPDAobjects');
0014 % Return required objects
0015 objs = objs(rows);
0016
0017 % END
```


ltpdv_hist_plot

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_hist_plot(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_hist_plot(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get selected objects
0008 objs = ltpdv_get_selected_objs(mainfig);
0009
0010 % Pass AOs to iplot
0011 for j=1:length(objs)
0012     obj = objs{j};
0013     if isa(obj, 'ao')
0014         plot(obj.hist);
0015     else
0016         warning('!!! Skipping object %s - it's not an AO.', obj.name);
0017     end
0018 end
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_importObjs.m

ltpdv_importObjs

PURPOSE [^]

LTPDV_IMPORTOBJJS import given objects to the object list

SYNOPSIS [^]

function ltpdv_importObjs(varargin)

DESCRIPTION [^]

```
LTPDV_IMPORTOBJJS import given objects to the object list
M Hewitson 22-04-08
$Id: ltpdv_importObjs.m,v 1.2 2008/06/18 15:07:35 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_importObjs(varargin)
0002 % LTPDV_IMPORTOBJJS import given objects to the object list
0003 %
0004 % M Hewitson 22-04-08
0005 %
0006 % $Id: ltpdv_importObjs.m,v 1.2 2008/06/18 15:07:35 hewitson Exp $
0007 %
0008
0009 myh      = varargin{1};
0010 mainfig = varargin{end-1};
0011 oh       = varargin{end};
0012
0013 % Get data
0014 data = get(oh, 'Data');
0015
0016 % Get selected rows
0017 rows = getappdata(mainfig, 'ImportWorkspaceSelectedRows');
0018
0019 % Do we rename to variable name?
0020 ch = findobj(mainfig, 'Tag', 'LTPDVusevarnameChk');
0021 if get(ch, 'Value')
0022     useVname = true;
0023 else
0024     useVname = false;
0025 end
0026
0027 % Data to import
0028 idata = data(rows,:);
0029 % Current objects
0030 % Get each object from the workspace
0031 for j=1:size(idata,1)
0032     % get variable name
0033     varname = idata{j,1};
0034     % get object
0035     obj = evalin('base', varname);
0036     if useVname
```

```
0037     for k=1:numel(obj)
0038         obj(k) = obj(k).setName(varname);
0039     end
0040 end
0041 % Add these objects to the object list
0042 for j=1:numel(obj)
0043     objs = getappdata(mainfig, 'LTPDAobjects');
0044     setappdata(mainfig, 'LTPDAobjects', [objs {obj(j)}]);
0045 end
0046 % Refresh object list
0047 ltpdv_refresh_object_list(mainfig);
0048 end
0049
0050
0051
0052 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_load_objects.m

ltpdv_load_objects

PURPOSE ^

Handles

SYNOPSIS ^

```
function ltpdv_load_objects(varargin)
```

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_load_objects(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get filename from user
0008 [filename, pathname] = uigetfile( ...
0009     {'*.mat', 'MATLAB MAT Files (*.mat)'; ...
0010     '*.xml', 'LTPDA XML Files (*.xml)'}; ...
0011     'Select a file');
0012
0013 if isequal(filename,0) || isequal(pathname,0)
0014     disp('Cancelled.')
0015 else
0016     fname = fullfile(pathname, filename);
0017
0018     % split filename up
0019     [pathstr, name, ext, vers] = fileparts(fname);
0020
0021     switch ext
0022     case '.mat'
0023
0024         obj = load(fname);
0025
0026     case '.xml'
0027
0028         root_node = xmlread(filename);
0029         obj = utils.helper.xmlread(root_node);
0030
0031     otherwise
0032         error('### Unsupported file format.');
```

```
0033     end
0034 end
0035
0036 % Add this object to the list
0037 objs = getappdata(mainfig, 'LTPDAobjects');
0038 setappdata(mainfig, 'LTPDAobjects', [objs {obj}]);
0039 % Refresh object list
0040 ltpdv_refresh_object_list(mainfig);
0041
0042
0043
```

0044 % END

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_objectList_edit.m

ltpdv_objectList_edit

PURPOSE [^]

LTPDV_OBJECTLIST_EDIT activates when the user changes the content of a

SYNOPSIS [^]

```
function ltpdv_objectList_edit(varargin)
```

DESCRIPTION [^]

LTPDV_OBJECTLIST_EDIT activates when the user changes the content of a cell.

M Hewitson 22-04-08

\$Id: ltpdv_objectList_edit.m,v 1.2 2008/06/18 15:07:35 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

SOURCE CODE [^]

```
0001 function ltpdv_objectList_edit(varargin)
0002 % LTPDV_OBJECTLIST_EDIT activates when the user changes the content of a
0003 % cell.
0004 %
0005 % M Hewitson 22-04-08
0006 %
0007 % $Id: ltpdv_objectList_edit.m,v 1.2 2008/06/18 15:07:35 hewitson Exp $
0008 %
0009
0010 % Handles
0011 myh      = varargin{1};
0012 mainfig = varargin{end};
0013
0014 % Structure of selection details
0015 sel = varargin{2};
0016
0017 % Get selected rows
0018 rows = unique(sel.Indices(:,1));
0019
0020 % Get current object list
0021 objs = getappdata(mainfig, 'LTPDAobjects');
0022
0023 objs{rows} = objs{rows}.setName(sel.NewData);
0024
0025 % Set objects back
0026 setappdata(mainfig, 'LTPDAobjects', objs);
0027
0028 % Refresh object list
0029 ltpdv_refresh_object_list(mainfig);
0030
0031
0032 % END
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_objectList_select.m

ltpdv_objectList_select

PURPOSE [^]

LTPDV_OBJECTLIST_SELECT activates when the user selects cells in the

SYNOPSIS [^]

```
function ltpdv_objectList_select(varargin)
```

DESCRIPTION [^]

LTPDV_OBJECTLIST_SELECT activates when the user selects cells in the workspace object list on the import panel

M Hewitson 22-04-08

\$Id: ltpdv_objectList_select.m,v 1.3 2008/09/02 07:51:47 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_properties_build_props](#) Object class

This function is called by:

SOURCE CODE [^]

```
0001 % LTPDV_OBJECTLIST_SELECT activates when the user selects cells in the
0002 % workspace object list on the import panel
0003 %
0004 % M Hewitson 22-04-08
0005 %
0006 % $Id: ltpdv_objectList_select.m,v 1.3 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008 function ltpdv_objectList_select(varargin)
0009
0010 % Handles
0011 myh = varargin{1};
0012 mainfig = varargin{end};
0013
0014 % Structure of selection details
0015 sel = varargin{2};
0016
0017 % Get selected rows
0018 rows = unique(sel.Indices(:,1));
0019 setappdata(mainfig, 'ObjectListSelectedRows', rows);
0020
0021 % Set tooltip
0022 objs = getappdata(mainfig, 'LTPDAobjects');
0023 if numel(rows) == 1
0024     w = display(objs{rows});
0025     str = [];
0026     for j=1:numel(w)
0027         str = [str sprintf('\n%s', w{j})];
0028     end
0029     set(myh, 'TooltipString', sprintf('%s', str));
0030
0031 % get object
0032 obj = objs{rows};
0033
0034     ltpdv_properties_build_props(obj, mainfig)
0035
0036 end
0037
0038
```

```
0039
0040 end
0041
0042 % END
```

ltpdv_parse_control_chans

PURPOSE ^

DV_PARSECONTROLCHANS parse out a list of control channels from the input

SYNOPSIS ^

function control_channels = dv_parse_control_chans(channels)

DESCRIPTION ^

DV_PARSECONTROLCHANS parse out a list of control channels from the input channel list.

Usage: control_channels = dv_parsecontrolchans(chanlist);

Input form:

channel = "top_level#sub_level_1#N#ch1#...#chN#sub_level_2#..."

Output form:

```
channels(1) = "top_level:sub_level_1:ch1"
channels(2) = "top_level:sub_level_1:ch2"
           ⋮
           ⋮
```

M Hewitson 26-07-06

\$Id: ltpdv_parse_control_chans.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_server_set_channel_list](#) Get normal channels

SOURCE CODE ^

```
0001 function control_channels = dv_parse_control_chans(channels)
0002
0003 % DV_PARSECONTROLCHANS parse out a list of control channels from the input
0004 % channel list.
0005 %
0006 % Usage: control_channels = dv_parsecontrolchans(chanlist);
0007 %
0008 % Input form:
0009 %
0010 % channel = "top_level#sub_level_1#N#ch1#...#chN#sub_level_2#..."
0011 %
0012 % Output form:
0013 %
0014 % channels(1) = "top_level:sub_level_1:ch1"
0015 % channels(2) = "top_level:sub_level_1:ch2"
0016 %           ⋮
0017 %           ⋮
0018 %
0019 %
0020 % M Hewitson 26-07-06
0021 %
0022 % $Id: ltpdv_parse_control_chans.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0023
0024
```

```

0025 % control data is sampled at 1Hz for now
0026 count = 1;
0027
0028 numtoplevels = length(channels(:,1));
0029
0030 ch = 1;
0031 control_channels = [];
0032 for top=1:numtoplevels
0033
0034     name = channels(top,:);
0035     if(length(name)>40)
0036
0037         % parse the input string
0038         % get first part as top level
0039         [toplevel, r] = strtok(name, '#');
0040         name = r;
0041         while(length(name) > 1)
0042
0043             % get sub level
0044             [sublevel, r] = strtok(name, '#');
0045             name = deblank(r);
0046
0047             % get the number of sub sub levels
0048             [strN, r] = strtok(name, '#');
0049             name = deblank(r);
0050             % convert strN
0051             N = eval(strN);
0052             % now get each sub sub level making channel
0053             % names as we go and extracting data as we go
0054             for s = 1:N
0055                 [subsublevel, r] = strtok(name, '#');
0056                 name = deblank(r);
0057                 control_channels = strvcats(control_channels,[toplevel ':' sublevel ':'
subsublevel ]);
0058                 ch = ch + 1;
0059             end
0060         end
0061     end
0062 end
0063
0064 ch = ch-1;
0065
0066
0067
0068 % END
0069

```

ltpdv_plot

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_plot(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.

This function is called by:

SOURCE CODE ^

```
0001 function ltpdv_plot(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get selected objects
0008 objs = ltpdv_get_selected_objs(mainfig);
0009
0010 % Pass AOs to iplot
0011 as = [];
0012 for j=1:length(objs)
0013     obj = objs{j}
0014     if isa(obj, 'ao')
0015         as = [as obj];
0016     else
0017         warning('!!! Skipping object %s - it''s not an AO.', obj.name);
0018     end
0019 end
0020
0021 if ~isempty(as)
0022     iplot(as)
0023 end
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_plotFilterResp.m

ltpdv_plotFilterResp

PURPOSE [^]

LTPDV_PLOTFILTERRESP plots the response of the selected filter.

SYNOPSIS [^]

```
function ltpdv_plotFilterResp(varargin)
```

DESCRIPTION [^]

```
LTPDV_PLOTFILTERRESP plots the response of the selected filter.
M Hewitson 24-04-08
$Id: ltpdv_plotFilterResp.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_filter_getf](#) get f1

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_plotFilterResp(varargin)
0002 % LTPDV_PLOTFILTERRESP plots the response of the selected filter.
0003 %
0004 % M Hewitson 24-04-08
0005 %
0006 % $Id: ltpdv_plotFilterResp.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0007 %
0008
0009 % Handles
0010 myh      = varargin{1};
0011 htab     = varargin{end-1};
0012 mainfig  = varargin{end};
0013
0014 % Get selected filter
0015 filts = getappdata(mainfig, 'Filters');
0016 % Get selection
0017 fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
0018 idx = get(fh, 'Value');
0019 sfilts = [];
0020 for j=1:length(filts)
0021     if ismember(j, idx)
0022         sfilts = [sfilts filts(j)];
0023     end
0024 end
0025
0026 % get frequency plist
0027 pl = ltpdv_filter_getf(mainfig);
0028
0029 % Get responses
0030 fr = [];
0031 for j=1:length(sfilts)
0032     filt = sfilts(j);
0033     disp(sprintf(' - filter %03d: %s', j, filt.name));
0034     r = resp(filt, pl);
0035     if isempty(fr)
0036         fr = r;
```



```
0037     else
0038         fr = fr.*r;
0039     end
0040 end
0041
0042 % Plot filter response
0043 iplot(fr);
0044
0045 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_poleAdd

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_poleAdd(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_pole_list](#) Get poles

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_poleAdd(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008
0009 % Get freq
0010 fh = findobj(htab, 'Tag', 'LTPDVfilterPZfreq');
0011 fstr = get(fh, 'String');
0012 if ~isempty(fstr)
0013     f = str2num(fstr);
0014
0015 % Get Q
0016 qh = findobj(htab, 'Tag', 'LTPDVfilterPZq');
0017 q = str2num(get(qh, 'String'));
0018
0019 % Add to pole list
0020 poles = getappdata(mainfig, 'FilterPoles');
0021 if isempty(q)
0022     poles = [poles pz(f)];
0023 else
0024     poles = [poles pz(f,q)];
0025 end
0026 setappdata(mainfig, 'FilterPoles', poles);
0027
0028 % Set pole list
0029 ltpdv_set_pole_list(mainfig);
0030 end
0031
0032
```


ltpdv_poleClear

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_poleClear(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_pole_list](#) Get poles

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_poleClear(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Clear list
0009 setappdata(mainfig, 'FilterPoles', []);
0010
0011 % Set pole list
0012 ltpdv_set_pole_list(mainfig);
```


ltpdv_poleDelete

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_poleDelete(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_pole_list](#) Get poles

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_poleDelete(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get selection
0009 lh = findobj(htab, 'Tag', 'LTPDVfilterPolelist');
0010 idx = get(lh, 'Value');
0011
0012 % Get poles
0013 poles = getappdata(mainfig, 'FilterPoles');
0014
0015 % Make new list
0016 ops = [];
0017 for j=1:length(poles)
0018     if ~ismember(j, idx)
0019         ops = [ops poles(j)];
0020     end
0021 end
0022 setappdata(mainfig, 'FilterPoles', ops);
0023
0024 % Set pole list
0025 ltpdv_set_pole_list(mainfig);
```


ltpdv_preproc_fcnlist

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_preproc_fcnlist(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_build_params_panel](#)

This function is called by:

 [ltpdv_buildPreprocessPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_preproc_fcnlist(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 paramh   = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get the selected method
0009 lidx = get(myh, 'Value');
0010 lstr = get(myh, 'String');
0011 fcn  = lstr{lidx};
0012
0013 % Get parameters for this method
0014 ii = ao.getInfo(fcn);
0015
0016 % Build params panel
0017 if numel(ii.plists) == 1
0018     ltpdv_build_params_panel(paramh, ii.plists);
0019 else
0020     ltpdv_build_params_panel(paramh, ii);
0021 end
0022
0023
0024
```


ltpdv_preproc_param_act

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_preproc_param_act(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_build_params_panel](#)

SOURCE CODE ^

```
0001 function ltpdv_preproc_param_act(varargin)
0002
0003
0004 % Handles
0005 myh      = varargin{1};
0006 panh     = varargin{end};
0007
0008 % Parameter panel children handles
0009 pancomps = getappdata(panh, 'pancomps');
0010
0011 % My state
0012 val = get(myh, 'Value');
0013
0014 % My line
0015 ml = find(myh==[pancomps{:,3}]);
0016
0017 if val
0018     set(pancomps{ml,2}, 'Enable', 'on');
0019 else
0020     set(pancomps{ml,2}, 'Enable', 'off');
0021 end
0022
0023
0024
0025
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_preproc_sets_callback.m

ltpdv_preproc_sets_callback

PURPOSE [^]

SYNOPSIS [^]

```
function ltpdv_preproc_sets_callback(varargin)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_build_params_panel](#)

This function is called by:

 [ltpdv_build_params_panel](#)

SOURCE CODE [^]

```
0001 function ltpdv_preproc_sets_callback(varargin)
0002
0003     myh = varargin{1};
0004     info = varargin{end};
0005
0006     seti = get(myh, 'Value');
0007
0008     plist = info.plists(seti);
0009
0010     ltpdv_build_params_panel(get(myh, 'Parent'), plist);
0011
0012
0013 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_properties_apply.m

ltpdv_properties_apply

PURPOSE [^]

LTPDV_PROPERTIES_APPLY callback for apply button on the object properties

SYNOPSIS [^]

function ltpdv_properties_apply(varargin)

DESCRIPTION [^]

LTPDV_PROPERTIES_APPLY callback for apply button on the object properties panel.

M Hewitson 26-08-08

\$Id: ltpdv_properties_apply.m,v 1.1 2008/09/02 07:51:47 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

 [ltpdv_properties_build_props](#) Object class

SOURCE CODE [^]

```

0001 % LTPDV_PROPERTIES_APPLY callback for apply button on the object properties
0002 % panel.
0003 %
0004 % M Hewitson 26-08-08
0005 %
0006 % $Id: ltpdv_properties_apply.m,v 1.1 2008/09/02 07:51:47 hewitson Exp $
0007 %
0008
0009 function ltpdv_properties_apply(varargin)
0010
0011 % Handles
0012 myh = varargin{1};
0013 oldVals = varargin{end-2};
0014 obj = varargin{end-1};
0015 mainfig = varargin{end};
0016 panh = get(myh, 'Parent');
0017
0018 % go through each label and edit box in order
0019 lhs = findobj(panh, 'Style', 'text');
0020 ehs = findobj(panh, 'Style', 'edit');
0021
0022 for kk=numel(lhs):-1:1
0023     fcn = get(lhs(kk), 'String');
0024     val = get(ehs(kk), 'String');
0025
0026     % check the value has changed
0027     ov = oldVals{numel(lhs)-kk+1};
0028     if ~strcmp(val, ov)
0029         feval(fcn, obj, val);
0030     end
0031 end
0032
0033 % Refresh object list
0034 ltpdv_refresh_object_list(mainfig);
0035

```

```
0036   display(obj)
0037 end
```

ltpdv_properties_build_props

PURPOSE ^

Object class

SYNOPSIS ^

function ltpdv_properties_build_props(obj, mainfig)

DESCRIPTION ^

Object class

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_properties_apply](#) LTPDV_PROPERTIES_APPLY callback for apply button on the object properties

This function is called by:

 [ltpdv_objectList_select](#) LTPDV_OBJECTLIST_SELECT activates when the user selects cells in the

SOURCE CODE ^

```
0001 function ltpdv_properties_build_props(obj, mainfig)
0002
0003     % Object class
0004     cl = class(obj);
0005
0006     % Get setter methods
0007     ms = methods(cl);
0008     setters = ms(strmatch('set', ms));
0009
0010     % Draw properties panel
0011     panh = findobj(mainfig, 'Tag', 'LTPDVobjectProperties');
0012     delete(get(panh, 'Children'))
0013
0014     pmarg = 0.025;
0015     keyl = pmarg;
0016     keyh = 0.055;
0017     keyw = 0.3;
0018     keyb = 1-pmarg-keyh;
0019     vall = pmarg*2 + keyw;
0020     valw = 0.5;
0021     oldVals = {};
0022     for kk=1:numel(setters)
0023
0024         sfcn = setters{kk};
0025         prop = lower(strrep(sfcn, 'set', ''));
0026         val = [];
0027         try
0028             val = obj.(prop);
0029         end
0030         if isa(val, 'ltpda_obj')
0031             val = char(val);
0032         end
0033         if isa(val, 'sym')
0034             val = char(val);
0035         end
0036         if ~isnumeric(val)
0037             oldVals = [oldVals {val}];
0038         % Draw label
```



```

0039     sth = uicontrol(panh, 'Style', 'text', ...
0040         'String', sfcn, ...
0041         'Units', 'normalized', ...
0042         'BackgroundColor', 'w', ...
0043         'FontSize', 12, ...
0044         'Position', [keyl keyb keyw keyh]);
0045
0046     % Draw edit box
0047     eh = uicontrol(panh, 'Style', 'edit', ...
0048         'String', val, ...
0049         'units', 'normalized', ...
0050         'BackgroundColor', 'w', ...
0051         'FontSize', 12, ...
0052         'Enable', 'on', ...
0053         'Position', [vall keyb valw keyh]);
0054
0055     keyb = keyb - pmarg - keyh;
0056 end
0057 end
0058 % Draw apply button
0059 pbh = uicontrol(panh, 'Style', 'pushbutton', ...
0060     'String', 'Apply', ...
0061     'Callback', {@ltpdv_properties_apply, oldVals, obj, mainfig}, ...
0062     'Units', 'normalized', ...
0063     'FontSize', 12, ...
0064     'Position', [pmarg pmarg 0.15 0.05]);
0065
0066 end

```

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_pzmResp.m

ltpdv_pzmResp

PURPOSE [^]

Handles

SYNOPSIS [^]

```
function ltpdv_pzmResp(varargin)
```

DESCRIPTION [^]

Handles

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_filter_get_pzmodel](#) Get poles

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_pzmResp(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig = varargin{end};
0007
0008 % Get pzmodel
0009 pzm = ltpdv_filter_get_pzmodel(mainfig);
0010
0011 % Make response
0012 iplot(resp(pzm))
0013
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_pzmStore

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_pzmStore(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_filter_get_pzmodel](#) Get poles
-  [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

-  [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_pzmStore(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get pzmodel
0009 pzm = ltpdv_filter_get_pzmodel(mainfig);
0010
0011 % Add to objects
0012 objs = getappdata(mainfig, 'LTPDAobjects');
0013 objs = [objs {pzm}];
0014 setappdata(mainfig, 'LTPDAobjects', objs);
0015
0016 % Refresh object list
0017 ltpdv_refresh_object_list(mainfig);
```


ltpdv_pzmStoreResp

PURPOSE ^

Handles

SYNOPSIS ^

```
function ltpdv_pzmStoreResp(varargin)
```

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_filter_get_pzmodel](#) Get poles
-  [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

-  [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_pzmStoreResp(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get pzmodel
0009 pzm = ltpdv_filter_get_pzmodel(mainfig);
0010
0011 % Add to objects
0012 objs = getappdata(mainfig, 'LTPDAobjects');
0013 objs = [objs {resp(pzm)}];
0014 setappdata(mainfig, 'LTPDAobjects', objs);
0015
0016 % Refresh object list
0017 ltpdv_refresh_object_list(mainfig);
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_refresh_object_list.m

ltpdv_refresh_object_list

PURPOSE ^

LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

SYNOPSIS ^

```
function ltpdv_refresh_object_list(mainfig)
```

DESCRIPTION ^

```
LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.
M Hewitson 22-04-08
$Id: ltpdv_refresh_object_list.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_build_buildobject](#) LTPDV_BUILD_BUILDOBJECT callback for the build object button on the build
-  [ltpdv_calc_btn](#) LTPDV_CALC_BTN called when one of the calculator buttons is pushed.
-  [ltpdv_delete_objects](#) LTPDV_DELETE_OBJECTS delete the selected objects from the object list
-  [ltpdv_fcn_run](#) LTPDV_FCN_RUN runs the command on the selected panel
-  [ltpdv_filterApply](#) Handles
-  [ltpdv_importObjs](#) LTPDV_IMPORTOBJJS import given objects to the object list
-  [ltpdv_load_objects](#) Handles
-  [ltpdv_objectList_edit](#) LTPDV_OBJECTLIST_EDIT activates when the user changes the content of a
-  [ltpdv_properties_apply](#) LTPDV_PROPERTIES_APPLY callback for apply button on the object properties
-  [ltpdv_pzmStore](#) Handles
-  [ltpdv_pzmStoreResp](#) Handles
-  [ltpdv_server_import](#) Handles
-  [ltpdv_storeFilter](#) LTPDV_STOREFILTER stores the current filter as an object in the object
-  [ltpdv_storeFilterResp](#) LTPDV_STOREFILTERRESP stores the response on the current filter in the

SOURCE CODE ^

```
0001 function ltpdv_refresh_object_list(mainfig)
0002
0003 % LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.
0004 %
0005 % M Hewitson 22-04-08
0006 %
0007 % $Id: ltpdv_refresh_object_list.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 objs = getappdata(mainfig, 'LTPDAobjects');
0011
0012 % Go through object array
0013 data = {};
0014 for j=1:length(objs)
```



```
0015     obj = objs{j};
0016     if isfield(obj, 'data')
0017         data = [data; {obj.name, [class(obj) '/' class(obj.data)]}];
0018     else
0019         data = [data; {obj.name, class(obj)}];
0020     end
0021 end
0022
0023 % Get list handle
0024 lh = findobj(mainfig, 'Tag', 'LTPDVobjList');
0025 set(lh, 'Data', data);
0026
0027
0028 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_save_objects.m

ltpdv_save_objects

PURPOSE [^]

Handles

SYNOPSIS [^]

```
function ltpdv_save_objects(varargin)
```

DESCRIPTION [^]

Handles

CROSS-REFERENCE INFORMATION [^]

This function calls:



[ltpdv_get_selected_objs](#) LTPDV_GET_SELECTED_OBJS gets the selected objects from the object list.

This function is called by:

SOURCE CODE [^]

```
0001 function ltpdv_save_objects(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get selected objects
0008 objs = ltpdv_get_selected_objs(mainfig);
0009
0010 % Get filename from user
0011 [filename, pathname] = uiputfile( ...
0012     {'*.mat','MAT-files (*.mat)'; ...
0013     '*.xml','LTPDA-XML-files (*.xml)'}, ...
0014     'Save as')
0015
0016
0017 if isequal(filename,0) || isequal(pathname,0)
0018     disp('Cancelled.')
0019 else
0020     fname = fullfile(pathname, filename);
0021     % split filename up
0022     [pathstr, name, ext, vers] = fileparts(fname);
0023
0024     switch ext
0025     case '.mat'
0026
0027         save(fname, 'objs');
0028
0029     case '.xml'
0030
0031         % Save each object
0032         for j=1:numel(objs)
0033             obj = objs{j};
0034
0035             % filename
0036             fname = fullfile(pathstr, sprintf('%s_%03d%s', name, j, ext));
0037
0038             % save object
0039             save(obj, fname)
0040
0041         end
0042     end
0043 end
```

```
0042
0043     end
0044
0045     otherwise
0046         error('### Unsupported file format. ');
0047     end
0048
0049
0050
0051 end
0052
0053
0054
0055
0056
0057 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_search_channels.m

ltpdv_search_channels

PURPOSE [^]

LTPDV_SEARCH_CHANNELS search the channel list and reduce it to those that

SYNOPSIS [^]

function ltpdv_search_channels(varargin)

DESCRIPTION [^]

LTPDV_SEARCH_CHANNELS search the channel list and reduce it to those that match the search criteria.

M Hewitson 26-07-06

\$Id: ltpdv_search_channels.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_server_get_latest_chans](#) Get server details
-  [ltpdv_server_set_channel_list](#) Get normal channels

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_search_channels(varargin)
0002
0003 % LTPDV_SEARCH_CHANNELS search the channel list and reduce it to those that
0004 % match the search criteria.
0005 %
0006 % M Hewitson 26-07-06
0007 %
0008 % $Id: ltpdv_search_channels.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0009
0010 % Handles
0011 myh = varargin{1};
0012 mainfig = varargin{end};
0013
0014 % channels
0015 ch = findobj(mainfig, 'Tag', 'LTPDVchannelList');
0016 chans = get(ch, 'String');
0017
0018 nchans = size(chans,1);
0019 chansout = [];
0020
0021 % First tokenize the search string
0022 sh = findobj(mainfig, 'Tag', 'LTPDVchannelSearchEdit');
0023 sstr = get(sh, 'String');
0024 curr = upper(sstr);
0025 if isempty(curr)
0026     [latest, chans] = ltpdv_server_get_latest_chans(mainfig);
0027     % Set channel list
0028     ltpdv_server_set_channel_list(mainfig, chans);
0029 else
0030
0031     [t, r] = strtok(curr);
0032     cstr = [];
0033     cstr = [cstr; cellstr(t)];
```

```
0034 while ~isempty(r)
0035     [t, r] = strtok(r);
0036     cstr = [cstr; cellstr(t)];
0037 end
0038
0039 if strcmp(curr, '')
0040     chansout = chans;
0041 else
0042     for str=1:length(cstr)
0043         for c=1:nchans
0044             idx = strfind(upper(chans(c,:)), char(cstr(str)));
0045             if ~isempty(idx)
0046                 chansout = strvcat(chansout, deblank(chans(c,:)));
0047             end
0048         end
0049     end
0050 end
0051
0052 if ~isempty(chansout)
0053     chansout = char(sort(cellstr(chansout)));
0054 end
0055
0056 ltpdv_server_set_channel_list(mainfig, chansout);
0057
0058 end
0059
0060
0061 % END
```

ltpdv_server_addTime

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_addTime(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_server_get_comment](#) Get handle
-  [ltpdv_server_get_start_time](#) Handle
-  [ltpdv_server_get_stop_time](#) Handle
-  [ltpdv_server_set_timeslist](#) Get times list

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_addTime(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig  = varargin{end};
0006
0007 % Add to times list
0008 tlist = getappdata(mainfig, 'TimesList');
0009
0010 % Build this time
0011 start = ltpdv_server_get_start_time(mainfig);
0012 stop  = ltpdv_server_get_stop_time(mainfig);
0013 comm  = ltpdv_server_get_comment(mainfig);
0014
0015
0016 if start > 0 && stop > 0
0017     t = {start, stop, comm};
0018
0019     tlist = [tlist; t];
0020
0021     setappdata(mainfig, 'TimesList', tlist);
0022
0023     ltpdv_server_set_timeslist(mainfig);
0024 end
0025
```


ltpdv_server_channellist

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_channellist(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_server_get_latest_chans](#) Get server details
-  [ltpdv_server_get_server](#) Get data type
-  [ltpdv_server_get_start_time](#) Handle
-  [ltpdv_server_get_stop_time](#) Handle

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_channellist(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get selected channels
0008 idx = get(myh, 'Value');
0009 if length(idx) == 1
0010
0011     % get channel
0012     strs = cellstr(get(myh, 'String'));
0013     channel = strs{idx};
0014
0015     % Get one second of data
0016     start = ltpdv_server_get_start_time(mainfig);
0017     stop = ltpdv_server_get_stop_time(mainfig);
0018     % Get server details
0019     [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);
0020
0021     switch dtype
0022     case 'frame data'
0023         pltype = 'FRAMES';
0024     case 'h(t) frames'
0025         pltype = 'HFRAMES';
0026     case 'raw data'
0027         pltype = 'RAW';
0028     case 'hour trends'
0029         pltype = 'HOURL';
0030     case 'day trends'
0031         pltype = 'DAY';
0032     case 'chour trends'
0033         pltype = 'CHOUR';
0034     case 'cdays trends'
0035         pltype = 'CDAY';
```



```
0036     end
0037
0038     if start < 0
0039         % get latest
0040         [start, chans] = ltpdv_server_get_latest_chans(mainfig);
0041     end
0042
0043     % get A0
0044     a = geo2ao(plist('Type', pltype, 'SERVER', server, 'port', port, 'channels', channel,
0045                     'starttime', start, 'nsecs', 1, 'STAT', stat));
0046
0047 else
0048
0049 end
0050
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_server_clearTimes

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_clearTimes(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_server_set_timeslist](#) Get times list

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_clearTimes(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Clear list
0008 setappdata(mainfig, 'TimesList', {});
0009 ltpdv_server_set_timeslist(mainfig)
```


ltpdv_server_datatype_select

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_datatype_select(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_server_server_select](#) Handles

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_datatype_select(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get data type
0008 idx = get(myh, 'Value');
0009 strs = get(myh, 'String');
0010 dtype = strs{idx};
0011
0012 % port field
0013 porth = findobj(mainfig, 'Tag', 'LTPDVserverPortEdit');
0014 % rds field
0015 rdsh = findobj(mainfig, 'Tag', 'LTPDVserverRdsEdit');
0016 % cal field
0017 calh = findobj(mainfig, 'Tag', 'LTPDVserverCalEdit');
0018 % stat field
0019 stath = findobj(mainfig, 'Tag', 'LTPDVserverStat');
0020 % server select
0021 ssh = findobj(mainfig, 'Tag', 'LTPDVserverServer');
0022
0023 switch dtype
0024     case 'frame data'
0025         set(porth, 'Enable', 'on');
0026         set(rdsh, 'Enable', 'off');
0027         set(calh, 'Enable', 'off');
0028         set(stath, 'Enable', 'off');
0029         set(ssh, 'Value', 1, 'String', {'localhost/9000', '130.75.117.159/9000',
0030 '130.75.117.73/9000'});
0031     case 'h(t) frames'
0032         set(porth, 'Enable', 'on');
0033         set(rdsh, 'Enable', 'on');
0034         set(calh, 'Enable', 'on');
0035         set(stath, 'Enable', 'off');
0036         set(ssh, 'Value', 1, 'String', {'localhost/9000', '130.75.117.159/9008',
0037 '130.75.117.73/9007', '130.75.117.73/9008'});
0038         set(rdsh, 'String', '9');
```

```
0039     set(calh, 'String', '1');
0040
0041     case 'raw data'
0042         set(porth, 'Enable', 'off');
0043         set(rdsh, 'Enable', 'off');
0044         set(calh, 'Enable', 'off');
0045         set(stath, 'Enable', 'off');
0046         set(ssh, 'Value', 1, 'String', {'localhost', '130.75.117.159'});
0047     case 'hour trends'
0048         set(porth, 'Enable', 'on');
0049         set(rdsh, 'Enable', 'off');
0050         set(calh, 'Enable', 'off');
0051         set(stath, 'Enable', 'on');
0052         set(ssh, 'Value', 1, 'String', {'localhost/9000', '130.75.117.159/9000',
'130.75.117.159/9008', '130.75.117.73/9007', '130.75.117.73/9008'});
0053     case 'day trends'
0054         set(porth, 'Enable', 'on');
0055         set(rdsh, 'Enable', 'off');
0056         set(calh, 'Enable', 'off');
0057         set(stath, 'Enable', 'on');
0058         set(ssh, 'Value', 1, 'String', {'localhost/9000', '130.75.117.159/9000',
'130.75.117.159/9008', '130.75.117.73/9007', '130.75.117.73/9008'});
0059     end
0060
0061
0062 ltpdv_server_server_select(0, mainfig);
```

ltpdv_server_deleteTimes

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_deleteTimes(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_deleteTimes(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007
0008
```


ltpdv_server_getChannels

PURPOSE [^]

SYNOPSIS [^]

function ltpdv_server_getChannels(varargin)

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_server_get_latest_chans](#) Get server details
-  [ltpdv_server_set_channel_list](#) Get normal channels

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_server_getChannels(varargin)
0002
0003 myh = varargin{1};
0004 mainfig = varargin{end};
0005
0006 [latest, chans] = ltpdv_server_get_latest_chans(mainfig);
0007 % Set channel list
0008 ltpdv_server_set_channel_list(mainfig, chans);
```


ltpdv_server_getLatest

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_getLatest(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_server_get_server](#) Get data type
-  [ltpdv_server_set_duration](#) Get handle

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_getLatest(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get server details
0008 [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);
0009
0010 switch dtype
0011     case 'frame data'
0012         latest = m2fserv(server, port, 'MIN');
0013     case 'raw data'
0014         latest = m2dserv(server);
0015     case 'h(t) frames'
0016         latest = m2fserv(server, port, 'HMIN');
0017     case 'hour trends'
0018         latest = m2fserv(server, port, 'HOUR');
0019     case 'day trends'
0020         latest = m2fserv(server, port, 'DAY');
0021     otherwise
0022         error('### Unknown data type');
0023 end
0024
0025 % Set start and stop times
0026 sh = findobj(mainfig, 'Tag', 'LTPDVserverStartEdit');
0027 set(sh, 'String', GPS2UTC(latest));
0028 sh = findobj(mainfig, 'Tag', 'LTPDVserverStopEdit');
0029 set(sh, 'String', GPS2UTC(latest));
0030 ltpdv_server_set_duration(mainfig, latest, latest)
```


ltpdv_server_get_channels

PURPOSE ^

Get handle to list

SYNOPSIS ^

```
function chans = ltpdv_server_get_channels(mainfig)
```

DESCRIPTION ^

Get handle to list

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_server_get_data](#) Get times

SOURCE CODE ^

```
0001 function chans = ltpdv_server_get_channels(mainfig)
0002
0003 % Get handle to list
0004 ch = findobj(mainfig, 'Tag', 'LTPDVchannelList');
0005 idx = get(ch, 'Value');
0006 str = cellstr(get(ch, 'String'));
0007
0008 chans = str(idx);
0009
```


ltpdv_server_get_comment

PURPOSE ^

Get handle

SYNOPSIS ^

```
function comment = ltpdv_server_get_comment(mainfig)
```

DESCRIPTION ^

Get handle

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_server_addTime](#) Handles
-  [ltpdv_server_get_times](#) Do we have a list?

SOURCE CODE ^

```
0001 function comment = ltpdv_server_get_comment(mainfig)
0002
0003 % Get handle
0004 ch = findobj(mainfig, 'Tag', 'LTPDVserverCommentEdit');
0005
0006 comment = get(ch, 'String');
0007
0008
0009
0010
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_server_get_data.m

ltpdv_server_get_data

PURPOSE [^]

Get times

SYNOPSIS [^]

```
function a = ltpdv_server_get_data(mainfig)
```

DESCRIPTION [^]

Get times

CROSS-REFERENCE INFORMATION [^]

This function calls:

-  [ltpdv_server_get_channels](#) Get handle to list
-  [ltpdv_server_get_server](#) Get data type
-  [ltpdv_server_get_times](#) Do we have a list?

This function is called by:

-  [ltpdv_server_import](#) Handles

SOURCE CODE [^]

```
0001 function a = ltpdv_server_get_data(mainfig)
0002
0003 % Get times
0004 times = ltpdv_server_get_times(mainfig);
0005
0006 % Get server details
0007 [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);
0008 spl = plist('Type', dtype, 'SERVER', server, 'port', port);
0009 % Get selected channels
0010 chans = ltpdv_server_get_channels(mainfig);
0011
0012 % Loop over times and channels
0013 a = [];
0014 for k=1:length(times)
0015     ts = times(k);
0016     start = char(ts.startT);
0017     nsecs = ts.endT-ts.startT+1;
0018     % Loop over channels
0019     for j=1:length(chans)
0020         chan = chans{j};
0021         disp(sprintf('=== Getting %s | %s [%ds]', chan, start, nsecs));
0022         na = geo2ao(spl, plist('channels', chan, 'starttime', start, 'nsecs', nsecs));
0023         a = [a na];
0024     end % End chans loop
0025 end % End time loop
0026
0027
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_server_get_latest_chans.m

ltpdv_server_get_latest_chans

PURPOSE [^]

Get server details

SYNOPSIS [^]

```
function [latest, chans] = ltpdv_server_get_latest_chans(mainfig)
```

DESCRIPTION [^]

Get server details

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_server_get_server](#) Get data type

This function is called by:

 [ltpdv_search_channels](#) LTPDV_SEARCH_CHANNELS search the channel list and reduce it to those that

 [ltpdv_server_channellist](#) Handles

 [ltpdv_server_getChannels](#)

 [ltpdv_server_queryServer](#) Handles

SOURCE CODE [^]

```
0001 function [latest, chans] = ltpdv_server_get_latest_chans(mainfig)
0002
0003 % Get server details
0004 [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);
0005
0006 switch dtype
0007     case 'frame data'
0008
0009         latest = m2fserv(server, port, 'MIN');
0010         chans = m2fserv(server, port, latest, 'MIN');
0011
0012     case 'raw data'
0013
0014         latest = m2dserv(server);
0015         chans = m2dserv(server, latest);
0016
0017     case 'h(t) frames'
0018
0019         latest = m2fserv(server, port, 'HMIN');
0020         chans = m2fserv(server, port, latest, 'HMIN', cal, rds);
0021
0022     case 'hour trends'
0023
0024         latest = m2fserv(server, port, 'HOUR');
0025         chans = m2fserv(server, port, latest, 'HOUR');
0026
0027     case 'day trends'
0028
0029         latest = m2fserv(server, port, 'DAY');
0030         chans = m2fserv(server, port, latest, 'DAY');
0031
0032     otherwise
0033         error('### Unknown data type');
```

0034 [end](#)

ltpdv_server_get_server

PURPOSE ^

Get data type

SYNOPSIS ^

```
function [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig)
```

DESCRIPTION ^

Get data type

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_server_channellist](#) Handles
-  [ltpdv_server_getLatest](#) Handles
-  [ltpdv_server_get_data](#) Get times
-  [ltpdv_server_get_latest_chans](#) Get server details

SOURCE CODE ^

```
0001 function [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig)
0002
0003
0004 % Get data type
0005 sh = findobj(mainfig, 'Tag', 'LTPDVserverDataType');
0006 idx = get(sh, 'Value');
0007 strs = get(sh, 'String');
0008 dtype = strs{idx};
0009
0010 % Get server
0011 sh = findobj(mainfig, 'Tag', 'LTPDVserverServerEdit');
0012 server = get(sh, 'String');
0013
0014 % Get port
0015 sh = findobj(mainfig, 'Tag', 'LTPDVserverPortEdit');
0016 port = str2num(get(sh, 'String'));
0017
0018 % Get rds
0019 sh = findobj(mainfig, 'Tag', 'LTPDVserverRdsEdit');
0020 rds = str2num(get(sh, 'String'));
0021
0022 % Get cal
0023 sh = findobj(mainfig, 'Tag', 'LTPDVserverCalEdit');
0024 cal = str2num(get(sh, 'String'));
0025
0026 % Get stat
0027 sh = findobj(mainfig, 'Tag', 'LTPDVserverStat');
0028 idx = get(sh, 'Value');
0029 strs = get(sh, 'String');
0030 stat = strs{idx};
0031
0032
0033
0034
0035
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_server_get_start_time.m

ltpdv_server_get_start_time

PURPOSE ^

Handle

SYNOPSIS ^

```
function gps = ltpdv_server_get_start_time(mainfig)
```

DESCRIPTION ^

Handle

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_server_get_stop_time](#) Handle

This function is called by:

 [ltpdv_server_addTime](#) Handles

 [ltpdv_server_channellist](#) Handles

 [ltpdv_server_get_stop_time](#) Handle

 [ltpdv_server_get_times](#) Do we have a list?

 [ltpdv_server_start_edit](#) Handles

 [ltpdv_server_stop_edit](#) Handles

SOURCE CODE ^

```

0001 function gps = ltpdv_server_get_start_time(mainfig)
0002
0003 % Handle
0004 sh = findobj(mainfig, 'Tag', 'LTPDVserverStartEdit');
0005
0006 % Get string
0007 sstr = get(sh, 'String');
0008
0009
0010 if sstr(1) == '+'
0011     % get start
0012     start = ltpdv_server_get_stop_time(mainfig);
0013     gps = start + str2num(sstr(2:end)) -1;
0014     set(sh, 'String', GPS2UTC(gps));
0015     return
0016 end
0017 if sstr(1) == '-'
0018     % get start
0019     start = ltpdv_server_get_stop_time(mainfig);
0020     gps = start - str2num(sstr(2:end)) +1;
0021     set(sh, 'String', GPS2UTC(gps));
0022     return
0023 end
0024
0025 % Is this GPS?
0026 if ~isempty(sstr)
0027     try
0028         gps = eval(sstr);
0029         utc = GPS2UTC(gps);
0030     catch
0031         utc = sstr;
0032         gps = UTC2GPS(utc);

```

```
0033     end
0034 else
0035     gps = -1;
0036 end
0037
0038
0039
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_server_get_stop_time.m

ltpdv_server_get_stop_time

PURPOSE ^

Handle

SYNOPSIS ^

```
function gps = ltpdv_server_get_stop_time(mainfig)
```

DESCRIPTION ^

Handle

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_server_get_start_time](#) Handle

This function is called by:

 [ltpdv_server_addTime](#) Handles
 [ltpdv_server_channellist](#) Handles
 [ltpdv_server_get_start_time](#) Handle
 [ltpdv_server_get_times](#) Do we have a list?
 [ltpdv_server_start_edit](#) Handles
 [ltpdv_server_stop_edit](#) Handles

SOURCE CODE ^

```
0001 function gps = ltpdv_server_get_stop_time(mainfig)
0002
0003 % Handle
0004 sh = findobj(mainfig, 'Tag', 'LTPDVserverStopEdit');
0005
0006 % Get string
0007 sstr = get(sh, 'String');
0008
0009 if sstr(1) == '+'
0010     % get start
0011     start = ltpdv_server_get_start_time(mainfig);
0012     gps = start + str2num(sstr(2:end)) -1;
0013     set(sh, 'String', GPS2UTC(gps));
0014     return
0015 end
0016 if sstr(1) == '-'
0017     % get start
0018     start = ltpdv_server_get_start_time(mainfig);
0019     gps = start - str2num(sstr(2:end)) +1;
0020     set(sh, 'String', GPS2UTC(gps));
0021     return
0022 end
0023
0024 % Is this GPS?
0025 if ~isempty(sstr)
0026     try
0027         gps = eval(sstr);
0028         utc = GPS2UTC(gps);
0029     catch
0030         utc = sstr;
0031         gps = UTC2GPS(utc);
0032     end
```



```
0033 else
0034     gps = -1;
0035 end
0036
0037
0038
0039
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_server_get_times.m

ltpdv_server_get_times

PURPOSE ^

Do we have a list?

SYNOPSIS ^

```
function tss = ltpdv_server_get_times(mainfig)
```

DESCRIPTION ^

Do we have a list?

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_server_get_comment](#) Get handle
-  [ltpdv_server_get_start_time](#) Handle
-  [ltpdv_server_get_stop_time](#) Handle

This function is called by:

-  [ltpdv_server_get_data](#) Get times

SOURCE CODE ^

```
0001 function tss = ltpdv_server_get_times(mainfig)
0002
0003
0004 % Do we have a list?
0005 lh = findobj(mainfig, 'Tag', 'LTPDVtimesList');
0006 tlist = getappdata(mainfig, 'TimesList');
0007
0008 if isempty(tlist)
0009
0010     % Get a single time from the entry boxes
0011     start = ltpdv_server_get_start_time(mainfig);
0012     stop = ltpdv_server_get_stop_time(mainfig);
0013
0014     if start < 0 || stop < 0
0015         error('### Please set a start and stop time or add some times to the list.');
```

Description of ltpdv_server_get_times

```
0038         end
0039         tss = [tss ts];
0040     end
0041
0042
0043 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpdv_server_import

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_import(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.
-  [ltpdv_server_get_data](#) Get times

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_import(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 a = ltpdv_server_get_data(mainfig);
0008
0009 % Add objects to the object list
0010
0011 % Get current objects
0012 objs = getappdata(mainfig, 'LTPDAobjects');
0013
0014 % Add these new ones
0015 for j=1:numel(a)
0016     objs = [objs {a(j)}];
0017 end
0018
0019 % Refresh object list
0020 setappdata(mainfig, 'LTPDAobjects', objs);
0021 ltpdv_refresh_object_list(mainfig);
0022
0023
```


ltpdv_server_queryServer

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_queryServer(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_server_get_latest_chans](#) Get server details
-  [ltpdv_server_set_channel_list](#) Get normal channels
-  [ltpdv_server_set_duration](#) Get handle

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_queryServer(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 [latest, chans] = ltpdv_server_get_latest_chans(mainfig);
0008
0009 % Set start and stop times
0010 sh = findobj(mainfig, 'Tag', 'LTPDVserverStartEdit');
0011 set(sh, 'String', GPS2UTC(latest));
0012 sh = findobj(mainfig, 'Tag', 'LTPDVserverStopEdit');
0013 set(sh, 'String', GPS2UTC(latest));
0014 ltpdv_server_set_duration(mainfig, latest, latest)
0015
0016 % Set channel list
0017 ltpdv_server_set_channel_list(mainfig, chans);
0018
```


ltpdv_server_server_select

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_server_server_select(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_buildImportPanel](#)
-  [ltpdv_server_datatype_select](#) Handles

SOURCE CODE ^

```
0001 function ltpdv_server_server_select(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get selected server
0008 sh      = findobj(mainfig, 'Tag', 'LTPDVserverServer');
0009 idx     = get(sh, 'Value');
0010 str     = get(sh, 'String');
0011 srv     = str{idx};
0012
0013 % split port off
0014 [server, port] = strtok(srv, '/');
0015 if ~isempty(port)
0016     port = str2num(port(2:end));
0017 end
0018
0019 % Set server edit box
0020 eh = findobj(mainfig, 'Tag', 'LTPDVserverServerEdit');
0021 set(eh, 'String', server);
0022
0023 % Set port edit box
0024 eh = findobj(mainfig, 'Tag', 'LTPDVserverPortEdit');
0025 set(eh, 'String', port);
0026
0027
0028
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_server_set_channel_list.m

ltpdv_server_set_channel_list

PURPOSE ^

Get normal channels

SYNOPSIS ^

```
function ltpdv_server_set_channel_list(mainfig, chans)
```

DESCRIPTION ^

Get normal channels

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_filter_control_chans](#) DV_FILTERCONTROLCHANS filter out control channels
-  [ltpdv_parse_control_chans](#) DV_PARSECONTROLCHANS parse out a list of control channels from the input

This function is called by:

-  [ltpdv_search_channels](#) LTPDV_SEARCH_CHANNELS search the channel list and reduce it to those that
-  [ltpdv_server_getChannels](#)
-  [ltpdv_server_queryServer](#) Handles

SOURCE CODE ^

```
0001 function ltpdv_server_set_channel_list(mainfig, chans)
0002
0003
0004 % Get normal channels
0005 channels = ltpdv_filter_control_chans(chans);
0006
0007 % Check is control checkbox
0008 ch = findobj(mainfig, 'Tag', 'LTPDVchannelIncludeControl');
0009 if get(ch, 'Value')
0010     % remove control chans
0011     control_chans = ltpdv_parse_control_chans(chans);
0012     channels = strvcats(control_chans, channels);
0013 end
0014
0015 % Get channel list
0016 ch = findobj(mainfig, 'Tag', 'LTPDVchannelList');
0017 set(ch, 'Value', 1);
0018 set(ch, 'String', channels);
0019
```


ltpdv_server_set_duration

PURPOSE ^

Get handle

SYNOPSIS ^

function ltpdv_server_set_duration(mainfig, start, stop)

DESCRIPTION ^

Get handle

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_server_getLatest](#) Handles
-  [ltpdv_server_queryServer](#) Handles
-  [ltpdv_server_start_edit](#) Handles
-  [ltpdv_server_stop_edit](#) Handles

SOURCE CODE ^

```
0001 function ltpdv_server_set_duration(mainfig, start, stop)
0002
0003 % Get handle
0004 dh = findobj(mainfig, 'Tag', 'LTPDVserverDurationDisplay');
0005
0006 % Get duration
0007 dur = stop - start + 1;
0008
0009 if dur < 0
0010     error('### The duration is negative. Change the start or stop time.');
```

```
0011 end
0012
0013 hours = fix(dur/3600);
0014 mins  = fix((dur-hours*3600)/60);
0015 secs  = dur-hours*3600-mins*60;
0016
0017 sign = '';
0018 if hours < 0
0019     sign = '-';
0020     hours = abs(hours);
0021 end
0022 if mins < 0
0023     sign = '-';
0024     mins = abs(mins);
0025 end
0026 if secs < 0
0027     sign = '-';
0028     secs = abs(secs);
0029 end
0030
0031 set(dh, 'String', sprintf('%s%02d:%02d:%02d', sign, hours, mins, secs));
0032
0033
```


ltpdv_server_set_timeslist

PURPOSE ^

Get times list

SYNOPSIS ^

function ltpdv_server_set_timeslist(mainfig)

DESCRIPTION ^

Get times list

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_server_addTime](#) Handles
-  [ltpdv_server_clearTimes](#) Handles

SOURCE CODE ^

```
0001 function ltpdv_server_set_timeslist(mainfig)
0002
0003 % Get times list
0004 tlist = getappdata(mainfig, 'TimesList');
0005
0006 % Get list handle
0007 lh = findobj(mainfig, 'Tag', 'LTPDVtimesList');
0008
0009 % Format list
0010 lstr = [];
0011 for j=1:size(tlist,1)
0012     t = tlist(j,:);
0013     lstr = [lstr sprintf('%s [%ds] / %s', GPS2UTC(t{1}), t{2}-t{1}+1, t{3})]];
0014 end
0015
0016 set(lh, 'Value', 1, 'String', lstr);
0017
```


ltpdv_server_start_edit

PURPOSE ^

Handles

SYNOPSIS ^

```
function ltpdv_server_start_edit(varargin)
```

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_server_get_start_time](#) Handle
-  [ltpdv_server_get_stop_time](#) Handle
-  [ltpdv_server_set_duration](#) Get handle

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_start_edit(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get start time
0008 start = ltpdv_server_get_start_time(mainfig);
0009 % Get stop time
0010 stop = ltpdv_server_get_stop_time(mainfig);
0011
0012 ltpdv_server_set_duration(mainfig, start, stop);
0013
```


ltpdv_server_stop_edit

PURPOSE ^

Handles

SYNOPSIS ^

```
function ltpdv_server_stop_edit(varargin)
```

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpdv_server_get_start_time](#) Handle
-  [ltpdv_server_get_stop_time](#) Handle
-  [ltpdv_server_set_duration](#) Get handle

This function is called by:

-  [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_server_stop_edit(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 mainfig = varargin{end};
0006
0007 % Get start time
0008 start = ltpdv_server_get_start_time(mainfig);
0009 % Get stop time
0010 stop = ltpdv_server_get_stop_time(mainfig);
0011
0012
0013 ltpdv_server_set_duration(mainfig, start, stop);
```


ltpdv_server_timeslist

PURPOSE [^]

SYNOPSIS [^]

```
function ltpdv_server_timeslist(varargin)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_server_timeslist(varargin)
```


ltpdv_set_filter_list

PURPOSE ^

Get filters

SYNOPSIS ^

function ltpdv_set_filter_list(mainfig)

DESCRIPTION ^

Get filters

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_filterAdd](#) Handles
-  [ltpdv_filterDelete](#) Handles
-  [ltpdv_filtersClear](#) Handles

SOURCE CODE ^

```
0001 function ltpdv_set_filter_list(mainfig)
0002
0003 % Get filters
0004 filts = getappdata(mainfig, 'Filters');
0005
0006 % Get handle to filter list
0007 fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
0008
0009 data = {};
0010 for j=1:length(filts)
0011     filt = filts(j);
0012     fc = find(filt.hist.plistUsed, 'fc');
0013     if ~isempty(fc)
0014         extra = mat2str(fc);
0015     else
0016         extra = '';
0017     end
0018     if isempty(extra)
0019         data = [data; {sprintf('%s', filt.name)}];
0020     else
0021         data = [data; {sprintf('%s | %s', filt.name, extra)}];
0022     end
0023 end
0024
0025 set(fh, 'String', data)
0026 set(fh, 'Value', 1)
0027
```


ltpdv_set_pole_list

PURPOSE ^

Get poles

SYNOPSIS ^

function ltpdv_set_pole_list(mainfig)

DESCRIPTION ^

Get poles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_poleAdd](#) Handles
-  [ltpdv_poleClear](#) Handles
-  [ltpdv_poleDelete](#) Handles

SOURCE CODE ^

```
0001 function ltpdv_set_pole_list(mainfig)
0002
0003 % Get poles
0004 poles = getappdata(mainfig, 'FilterPoles');
0005
0006 % Get handle to pole list
0007 lh = findobj(mainfig, 'Tag', 'LTPDVfilterPolelist');
0008 set(lh, 'Value', 1);
0009
0010 % Build string
0011 str = {};
0012 for j=1:length(poles)
0013     p = poles(j);
0014     if isnan(p.q)
0015         str = [str {num2str(p.f)}];
0016     else
0017         str = [str {[num2str(p.f) ' ' num2str(p.q)]]];
0018     end
0019 end
0020
0021 set(lh, 'String', str);
```


ltpdv_set_zero_list

PURPOSE ^

Get poles

SYNOPSIS ^

```
function ltpdv_set_zero_list(mainfig)
```

DESCRIPTION ^

Get poles

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

-  [ltpdv_zeroAdd](#) Handles
-  [ltpdv_zeroClear](#) Handles
-  [ltpdv_zeroDelete](#) Handles

SOURCE CODE ^

```
0001 function ltpdv_set_zero_list(mainfig)
0002
0003 % Get poles
0004 poles = getappdata(mainfig, 'FilterZeros');
0005
0006 % Get handle to pole list
0007 lh = findobj(mainfig, 'Tag', 'LTPDVfilterZerolist');
0008 set(lh, 'Value', 1);
0009
0010 % Build string
0011 str = {};
0012 for j=1:length(poles)
0013     p = poles(j);
0014     if isnan(p.q)
0015         str = [str {num2str(p.f)}];
0016     else
0017         str = [str {[num2str(p.f) ', ' num2str(p.q)]}];
0018     end
0019 end
0020
0021 set(lh, 'String', str);
```


ltpdv_spectral_fcnlist

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_spectral_fcnlist(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_build_params_panel](#)

This function is called by:

 [ltpdv_buildSpectralPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_spectral_fcnlist(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 paramh   = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get the selected method
0009 lidc = get(myh, 'Value');
0010 lstr = get(myh, 'String');
0011 fcn  = lstr{lidx};
0012
0013 % Get parameters for this method
0014 if ismethod(ao, fcn)
0015     ii = ao.getInfo(fcn);
0016     pl = ii.plist(1);
0017 else
0018     pl = feval(fcn, 'Params');
0019 end
0020
0021 % Build params panel
0022 ltpdv_build_params_panel(paramh, pl);
0023
0024
0025
0026
```


ltpdv_storeFilter

PURPOSE ^

LTPDV_STOREFILTER stores the current filter as an object in the object

SYNOPSIS ^

function ltpdv_storeFilter(varargin)

DESCRIPTION ^

LTPDV_STOREFILTER stores the current filter as an object in the object list.
M Hewitson 24-04-08
\$Id: ltpdv_storeFilter.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_storeFilter(varargin)
0002 % LTPDV_STOREFILTER stores the current filter as an object in the object
0003 % list.
0004 %
0005 % M Hewitson 24-04-08
0006 %
0007 % $Id: ltpdv_storeFilter.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 % Handles
0011 myh      = varargin{1};
0012 htab     = varargin{end-1};
0013 mainfig  = varargin{end};
0014
0015 filts = getappdata(mainfig, 'Filters');
0016
0017 % Get selection
0018 fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
0019 idx = get(fh, 'Value');
0020 sfilts = [];
0021 for j=1:length(filts)
0022     if ismember(j, idx)
0023         sfilts = [sfilts filts(j)];
0024     end
0025 end
0026
0027 % Add filter to the object list
0028 objs = getappdata(mainfig, 'LTPDAobjects');
0029 for j=1:length(sfilts)
0030     filt = sfilts(j);
0031     objs = [objs {filt}];
0032 end
0033 setappdata(mainfig, 'LTPDAobjects', objs);
0034
0035 % Refresh object list
```

```
0036 ltpdv_refresh_object_list(mainfig);  
0037  
0038  
0039 % END
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_storeFilterResp.m

ltpdv_storeFilterResp

PURPOSE [^]

LTPDV_STOREFILTERRESP stores the response on the current filter in the

SYNOPSIS [^]

function ltpdv_storeFilterResp(varargin)

DESCRIPTION [^]

LTPDV_STOREFILTERRESP stores the response on the current filter in the object list.

M Hewitson 24-04-08

\$Id: ltpdv_storeFilterResp.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION [^]

This function calls:

 [ltpdv_filter_getf](#) get f1

 [ltpdv_refresh_object_list](#) LTPDV_REFRESH_OBJECT_LIST sets the object list from the app data.

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE [^]

```
0001 function ltpdv_storeFilterResp(varargin)
0002
0003 % LTPDV_STOREFILTERRESP stores the response on the current filter in the
0004 % object list.
0005 %
0006 % M Hewitson 24-04-08
0007 %
0008 % $Id: ltpdv_storeFilterResp.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0009 %
0010
0011 % Handles
0012 myh      = varargin{1};
0013 htab     = varargin{end-1};
0014 mainfig  = varargin{end};
0015
0016 % Get selected filter
0017 filts = getappdata(mainfig, 'Filters');
0018 % Get selection
0019 fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
0020 idx = get(fh, 'Value');
0021 sfilts = [];
0022 for j=1:length(filts)
0023     if ismember(j, idx)
0024         sfilts = [sfilts filts(j)];
0025     end
0026 end
0027
0028 % get frequency plist
0029 pl = ltpdv_filter_getf(mainfig);
0030
0031 % Get responses
0032 fr = [];
0033 for j=1:length(sfilts)
```



```
0034     filt = sfilts(j);
0035     disp(sprintf('    - filter %03d: %s', j, filt.name));
0036     r = resp(filt, pl);
0037     if isempty(fr)
0038         fr = r;
0039     else
0040         fr = fr.*r;
0041     end
0042 end
0043
0044 % Add filter to the object list
0045 objs = getappdata(mainfig, 'LTPDAobjects');
0046 setappdata(mainfig, 'LTPDAobjects', [objs {fr}]);
0047
0048 % Refresh object list
0049 ltpdv_refresh_object_list(mainfig);
0050
0051
0052
0053 % END
```

ltpdv_tabChanged

PURPOSE ^

SYNOPSIS ^

function ltpdv_tabChanged(varargin)

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_tabChanged(varargin)
0002
0003 myh    = varargin{1};
0004 edata  = varargin{2};
0005 % get old tab
0006 ti = edata.OldValue;
0007 tabs = get(myh, 'Children');
0008 if ti > 0
0009     oth = tabs(ti);
0010
0011     % Find all java objects below oth
0012     jobjs = findobj(oth, 'Type','uitabgroup');
0013     set(jobjs, 'Visible', 'off');
0014
0015 end
0016
0017 if ti > 0
0018     % get new tab
0019     ti = edata.NewValue;
0020     tabs = get(myh, 'Children');
0021     nth = tabs(ti);
0022
0023     % Find all java objects below oth
0024     jobjs = findobj(nth, 'Type','uitabgroup');
0025     set(jobjs, 'Visible', 'on');
0026 end
0027 drawnow
0028
0029 h = findobj(myh, 'Tag', 'LTPDVimportTabPanel');
0030 if h~=myh
0031     set(h, 'SelectedIndex', 1);
0032 end
0033
0034 end
```


ltpdv_workspaceList_select

PURPOSE ^

LTPDV_WORKSPACELIST_SELECT activates when the user selects cells in the

SYNOPSIS ^

function ltpdv_workspaceList_select(varargin)

DESCRIPTION ^

LTPDV_WORKSPACELIST_SELECT activates when the user selects cells in the workspace object list on the import panel

M Hewitson 22-04-08

\$Id: ltpdv_workspaceList_select.m,v 1.1 2008/05/11 10:38:43 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_workspaceList_select(varargin)
0002 % LTPDV_WORKSPACELIST_SELECT activates when the user selects cells in the
0003 % workspace object list on the import panel
0004 %
0005 % M Hewitson 22-04-08
0006 %
0007 % $Id: ltpdv_workspaceList_select.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 % Handles
0011 myh      = varargin{1};
0012 mainfig = varargin{end};
0013
0014 % Structure of selection details
0015 sel = varargin{2};
0016
0017 % Get selected rows
0018 rows = unique(sel.Indices(:,1));
0019 setappdata(mainfig, 'ImportWorkspaceSelectedRows', rows);
0020
0021 % END
```


[Home](#) > [m](#) > [gui](#) > [ltpdv](#) > [callbacks](#) > ltpdv_workspaceList_set_data.m

ltpdv_workspaceList_set_data

PURPOSE ^

LTPDV_WORKSPACELIST_SET_DATA sets the data into the workspace table

SYNOPSIS ^

function ltpdv_workspaceList_set_data(varargin)

DESCRIPTION ^

```
LTPDV_WORKSPACELIST_SET_DATA sets the data into the workspace table
M Hewitson 22-04-08
$Id: ltpdv_workspaceList_set_data.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
```

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_getWorkspaceObjs](#) LTPDV_GETWORKSPACEOBSJS gets a list of LTPDA user objects currently in the

This function is called by:

 [ltpdv_buildImportPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_workspaceList_set_data(varargin)
0002
0003 % LTPDV_WORKSPACELIST_SET_DATA sets the data into the workspace table
0004 %
0005 % M Hewitson 22-04-08
0006 %
0007 % $Id: ltpdv_workspaceList_set_data.m,v 1.1 2008/05/11 10:38:43 hewitson Exp $
0008 %
0009
0010 % Get panel handle
0011 th = varargin{end};
0012
0013 % Make a list of LTPDA workspace user objects
0014 objs = ltpdv_getWorkspaceObjs;
0015
0016 % Make a data cell array
0017 data = {};
0018 for j=1:length(objs)
0019     data = [data; {objs(j).name, objs(j).class, mat2str(objs(j).size),
0020 objs(j).bytes/1024}];
0021 end
0022 % Set the data
0023 set(th, 'Data', data);
0024 % END
```


ltpdv_zeroAdd

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_zeroAdd(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_zero_list](#) Get poles

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_zeroAdd(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008
0009 % Get freq
0010 fh = findobj(htab, 'Tag', 'LTPDVfilterPZfreq');
0011 fstr = get(fh, 'String');
0012 if ~isempty(fstr)
0013     f = str2num(fstr);
0014
0015 % Get Q
0016 qh = findobj(htab, 'Tag', 'LTPDVfilterPZq');
0017 q = str2num(get(qh, 'String'));
0018
0019 % Add to zero list
0020 zeros = getappdata(mainfig, 'FilterZeros');
0021 if isempty(q)
0022     zeros = [zeros pz(f)];
0023 else
0024     zeros = [zeros pz(f,q)];
0025 end
0026 setappdata(mainfig, 'FilterZeros', zeros);
0027
0028 % Set pole list
0029 ltpdv_set_zero_list(mainfig);
0030 end
0031
0032
```


ltpdv_zeroClear

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_zeroClear(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_zero_list](#) Get poles

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_zeroClear(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Clear list
0009 setappdata(mainfig, 'FilterZeros', []);
0010
0011 % Set pole list
0012 ltpdv_set_zero_list(mainfig);
```


ltpdv_zeroDelete

PURPOSE ^

Handles

SYNOPSIS ^

function ltpdv_zeroDelete(varargin)

DESCRIPTION ^

Handles

CROSS-REFERENCE INFORMATION ^

This function calls:

 [ltpdv_set_zero_list](#) Get poles

This function is called by:

 [ltpdv_buildFilterPanel](#)

SOURCE CODE ^

```
0001 function ltpdv_zeroDelete(varargin)
0002
0003 % Handles
0004 myh      = varargin{1};
0005 htab     = varargin{end-1};
0006 mainfig  = varargin{end};
0007
0008 % Get selection
0009 lh = findobj(htab, 'Tag', 'LTPDVfilterZerolist');
0010 idx = get(lh, 'Value');
0011
0012 % Get zeros
0013 zeros = getappdata(mainfig, 'FilterZeros');
0014
0015 % Make new list
0016 ops = [];
0017 for j=1:length(zeros)
0018     if ~ismember(j, idx)
0019         ops = [ops zeros(j)];
0020     end
0021 end
0022 setappdata(mainfig, 'FilterZeros', ops);
0023
0024 % Set pole list
0025 ltpdv_set_zero_list(mainfig);
```


Contents

PURPOSE ^

%%%%%%%%%%%% path: m/gui/pzmodel_designer
%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%%%%%%%% path: m/gui/pzmodel_designer %%%%%%%%%
 [m/gui/pzmodel_designer/pzmodel_helper](matlab:help m/gui/pzmodel_designer/pzmodel_helper) - M-file
 for pzmodel_helper.fig

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

[Home](#) > [m](#) > [gui](#) > [pzmodel_designer](#) > pzmodel_helper.m

pzmodel_helper

PURPOSE [^]

PZMODEL_HELPER M-file for pzmodel_helper.fig

SYNOPSIS [^]

function varargout = pzmodel_helper(varargin)

DESCRIPTION [^]

```
PZMODEL_HELPER M-file for pzmodel_helper.fig
PZMODEL_HELPER, by itself, creates a new PZMODEL_HELPER or raises the existing
singleton*.

H = PZMODEL_HELPER returns the handle to a new PZMODEL_HELPER or the handle to
the existing singleton*.

PZMODEL_HELPER('CALLBACK',hObject,eventData,handles,...) calls the local
function named CALLBACK in PZMODEL_HELPER.M with the given input arguments.

PZMODEL_HELPER('Property','Value',...) creates a new PZMODEL_HELPER or raises the
existing singleton*. Starting from the left, property value pairs are
applied to the GUI before pzmodel_helper_OpeningFunction gets called. An
unrecognized property name or invalid value makes property application
stop. All inputs are passed to pzmodel_helper_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

$Id: pzmodel_helper.m,v 1.10 2008/08/04 13:28:03 hewitson Exp $
```

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

-  [function pzmodel_helper_OpeningFcn\(hObject, eventdata, handles, varargin\)](#)
-  [function varargout = pzmodel_helper_OutputFcn\(hObject, eventdata, handles\)](#)
-  [function pzEdit_Callback\(hObject, eventdata, handles\)](#)
-  [function pzEdit_CreateFcn\(hObject, eventdata, handles\)](#)
-  [function addPoleBtn_Callback\(hObject, eventdata, handles\)](#)
-  [function poleList_Callback\(hObject, eventdata, handles\)](#)
-  [function poleList_CreateFcn\(hObject, eventdata, handles\)](#)
-  [function addZeroBtn_Callback\(hObject, eventdata, handles\)](#)
-  [function zeroList_Callback\(hObject, eventdata, handles\)](#)
-  [function zeroList_CreateFcn\(hObject, eventdata, handles\)](#)
-  [function updatePolesList\(handles\)](#)
-  [function updateZerosList\(handles\)](#)
-  [function updateModelResp\(handles\)](#)
-  [function gainEdit_Callback\(hObject, eventdata, handles\)](#)
-  [function gainEdit_CreateFcn\(hObject, eventdata, handles\)](#)

-  function f1Edit_Callback(hObject, eventdata, handles)
-  function f1Edit_CreateFcn(hObject, eventdata, handles)
-  function f2Edit_Callback(hObject, eventdata, handles)
-  function f2Edit_CreateFcn(hObject, eventdata, handles)
-  function nfEdit_Callback(hObject, eventdata, handles)
-  function nfEdit_CreateFcn(hObject, eventdata, handles)
-  function replotBtn_Callback(hObject, eventdata, handles)
-  function fsEdit_Callback(hObject, eventdata, handles)
-  function fsEdit_CreateFcn(hObject, eventdata, handles)
-  function figureBtn_Callback(hObject, eventdata, handles)
-  function filterStrEdit_Callback(hObject, eventdata, handles)
-  function filterStrEdit_CreateFcn(hObject, eventdata, handles)
-  function clearBtn_Callback(hObject, eventdata, handles)
-  function deletePolesBtn_Callback(hObject, eventdata, handles)
-  function deleteZerosBtn_Callback(hObject, eventdata, handles)
-  function saveAObtn_Callback(hObject, eventdata, handles)

SOURCE CODE ^

```

0001 function varargout = pzmodel_helper(varargin)
0002 % PZMODEL_HELPER M-file for pzmodel_helper.fig
0003 %     PZMODEL_HELPER, by itself, creates a new PZMODEL_HELPER or raises the existing
0004 %     singleton*.
0005 %
0006 %     H = PZMODEL_HELPER returns the handle to a new PZMODEL_HELPER or the handle to
0007 %     the existing singleton*.
0008 %
0009 %     PZMODEL_HELPER('CALLBACK',hObject,eventData,handles,...) calls the local
0010 %     function named CALLBACK in PZMODEL_HELPER.M with the given input arguments.
0011 %
0012 %     PZMODEL_HELPER('Property','Value',...) creates a new PZMODEL_HELPER or raises the
0013 %     existing singleton*. Starting from the left, property value pairs are
0014 %     applied to the GUI before pzmodel_helper_OpeningFunction gets called. An
0015 %     unrecognized property name or invalid value makes property application
0016 %     stop. All inputs are passed to pzmodel_helper_OpeningFcn via varargin.
0017 %
0018 %     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
0019 %     instance to run (singleton)".
0020 %
0021 % See also: GUIDE, GUIDATA, GUIHANDLES
0022 %
0023 % $Id: pzmodel_helper.m,v 1.10 2008/08/04 13:28:03 hewitson Exp $
0024 %
0025
0026
0027 % Edit the above text to modify the response to help pzmodel_helper
0028
0029 % Last Modified by GUIDE v2.5 19-Apr-2007 10:37:13
0030
0031 % Begin initialization code - DO NOT EDIT
0032 gui_Singleton = 1;
0033 gui_State = struct('gui_Name',       mfilename, ...
0034                   'gui_Singleton',   gui_Singleton, ...
0035                   'gui_OpeningFcn', @pzmodel_helper_OpeningFcn, ...
0036                   'gui_OutputFcn',  @pzmodel_helper_OutputFcn, ...
0037                   'gui_LayoutFcn',  [], ...
0038                   'gui_Callback',    []);
0039 if nargin && ischar(varargin{1})
0040     gui_State.gui_Callback = str2func(varargin{1});
0041 end
0042
0043 if nargout
0044     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
0045 else
0046     gui_mainfcn(gui_State, varargin{:});
0047 end
0048 % End initialization code - DO NOT EDIT
0049
0050
0051 % --- Executes just before pzmodel_helper is made visible.
0052 function pzmodel_helper_OpeningFcn(hObject, eventdata, handles, varargin)
0053 % This function has no output args, see OutputFcn.
0054 % hObject    handle to figure

```

```

0055 % eventdata reserved - to be defined in a future version of MATLAB
0056 % handles structure with handles and user data (see GUIDATA)
0057 % varargin command line arguments to pzmodel_helper (see VARARGIN)
0058
0059 % Choose default command line output for pzmodel_helper
0060 handles.output = hObject;
0061
0062 % Update handles structure
0063 guidata(hObject, handles);
0064
0065 % UIWAIT makes pzmodel_helper wait for user response (see UIRESUME)
0066 % uiwait(handles.main);
0067
0068 % create empty lists
0069 poles = [];
0070 zeros = [];
0071
0072 setappdata(handles.main, 'poles', poles);
0073 setappdata(handles.main, 'zeros', zeros);
0074 setappdata(handles.main, 'filt', []);
0075
0076 % --- Outputs from this function are returned to the command line.
0077 function varargout = pzmodel_helper_OutputFcn(hObject, eventdata, handles)
0078 % varargout cell array for returning output args (see VARARGOUT);
0079 % hObject handle to figure
0080 % eventdata reserved - to be defined in a future version of MATLAB
0081 % handles structure with handles and user data (see GUIDATA)
0082
0083 % Get default command line output from handles structure
0084 varargout{1} = handles.output;
0085
0086
0087
0088 function pzEdit_Callback(hObject, eventdata, handles)
0089 % hObject handle to pzEdit (see GCBO)
0090 % eventdata reserved - to be defined in a future version of MATLAB
0091 % handles structure with handles and user data (see GUIDATA)
0092
0093 % Hints: get(hObject,'String') returns contents of pzEdit as text
0094 % str2double(get(hObject,'String')) returns contents of pzEdit as a double
0095
0096
0097 % --- Executes during object creation, after setting all properties.
0098 function pzEdit_CreateFcn(hObject, eventdata, handles)
0099 % hObject handle to pzEdit (see GCBO)
0100 % eventdata reserved - to be defined in a future version of MATLAB
0101 % handles empty - handles not created until after all CreateFcns called
0102
0103 % Hint: edit controls usually have a white background on Windows.
0104 % See ISPC and COMPUTER.
0105 if ispc && isequal(get(hObject,'BackgroundColor'),
0106 get(0,'defaultUiControlBackgroundColor'))
0107 set(hObject,'BackgroundColor','white');
0108 end
0109
0110 % --- Executes on button press in addPoleBtn.
0111 function addPoleBtn_Callback(hObject, eventdata, handles)
0112 % hObject handle to addPoleBtn (see GCBO)
0113 % eventdata reserved - to be defined in a future version of MATLAB
0114 % handles structure with handles and user data (see GUIDATA)
0115
0116 % get values from pzEdit
0117 strVal = get(handles.pzEdit, 'String');
0118 val = str2num(strVal);
0119 f = val(1);
0120 if length(val) == 2
0121 q = val(2);
0122 p = pz(f,q);
0123 else
0124 p = pz(f);
0125 end
0126
0127 poles = getappdata(handles.main, 'poles');
0128 poles = [poles p];
0129 setappdata(handles.main, 'poles', poles);
0130
0131 % update list
0132 updatePolesList(handles);
0133
0134 % update model
0135 updateModelResp(handles);
0136
0137
0138 % --- Executes on selection change in poleList.
0139 function poleList_Callback(hObject, eventdata, handles)

```

```

0140 % hObject      handle to poleList (see GCBO)
0141 % eventdata    reserved - to be defined in a future version of MATLAB
0142 % handles      structure with handles and user data (see GUIDATA)
0143
0144 % Hints: contents = get(hObject,'String') returns poleList contents as cell array
0145 %             contents{get(hObject,'Value')} returns selected item from poleList
0146
0147
0148 % --- Executes during object creation, after setting all properties.
0149 function poleList_CreateFcn(hObject, eventdata, handles)
0150 % hObject      handle to poleList (see GCBO)
0151 % eventdata    reserved - to be defined in a future version of MATLAB
0152 % handles      empty - handles not created until after all CreateFcns called
0153
0154 % Hint: listbox controls usually have a white background on Windows.
0155 %             See ISPC and COMPUTER.
0156 if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
0157     set(hObject,'BackgroundColor','white');
0158 end
0159
0160
0161 % --- Executes on button press in addZeroBtn.
0162 function addZeroBtn_Callback(hObject, eventdata, handles)
0163 % hObject      handle to addZeroBtn (see GCBO)
0164 % eventdata    reserved - to be defined in a future version of MATLAB
0165 % handles      structure with handles and user data (see GUIDATA)
0166
0167 % get values from pzEdit
0168 strVal = get(handles.pzEdit, 'String');
0169 val     = str2num(strVal);
0170 f       = val(1);
0171 if length(val) == 2
0172     q = val(2);
0173     z = pz(f,q);
0174 else
0175     z = pz(f);
0176 end
0177
0178 zeros = getappdata(handles.main, 'zeros');
0179 zeros = [zeros z];
0180 setappdata(handles.main, 'zeros', zeros);
0181
0182 % update list
0183 updateZerosList(handles);
0184
0185 % update model
0186 updateModelResp(handles);
0187
0188
0189 % --- Executes on selection change in zeroList.
0190 function zeroList_Callback(hObject, eventdata, handles)
0191 % hObject      handle to zeroList (see GCBO)
0192 % eventdata    reserved - to be defined in a future version of MATLAB
0193 % handles      structure with handles and user data (see GUIDATA)
0194
0195 % Hints: contents = get(hObject,'String') returns zeroList contents as cell array
0196 %             contents{get(hObject,'Value')} returns selected item from zeroList
0197
0198
0199 % --- Executes during object creation, after setting all properties.
0200 function zeroList_CreateFcn(hObject, eventdata, handles)
0201 % hObject      handle to zeroList (see GCBO)
0202 % eventdata    reserved - to be defined in a future version of MATLAB
0203 % handles      empty - handles not created until after all CreateFcns called
0204
0205 % Hint: listbox controls usually have a white background on Windows.
0206 %             See ISPC and COMPUTER.
0207 if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
0208     set(hObject,'BackgroundColor','white');
0209 end
0210
0211 %----- Update Poles List -----
0212 function updatePolesList(handles)
0213
0214 poles = getappdata(handles.main, 'poles');
0215 pstr = [];
0216 for n=1:length(poles)
0217     p = poles(n);
0218     f = p.f;
0219     q = p.q;
0220     if q>0.5
0221         pstr = strvcat(pstr, sprintf('%2.2f Hz   Q=%2.2f', f, q));
0222     else
0223         pstr = strvcat(pstr, sprintf('%2.2f Hz', f));

```

```

0224     end
0225 end
0226
0227 if isempty(poles)
0228     set(handles.poleList, 'String', ' ');
0229 else
0230     set(handles.poleList, 'String', pstr);
0231 end
0232 set(handles.poleList, 'Value', 1);
0233
0234 %----- Update Zeros List -----
0235 function updateZerosList(handles)
0236
0237 zeros = getappdata(handles.main, 'zeros');
0238 pstr = [];
0239 for n=1:length(zeros)
0240     p = zeros(n);
0241     f = p.f;
0242     q = p.q;
0243     if q>0.5
0244         pstr = strvcats(pstr, sprintf('%2.2f Hz   Q=%2.2f', f, q));
0245     else
0246         pstr = strvcats(pstr, sprintf('%2.2f Hz', f));
0247     end
0248 end
0249
0250 if isempty(zeros)
0251     set(handles.zeroList, 'String', ' ');
0252 else
0253     set(handles.zeroList, 'String', pstr);
0254 end
0255 set(handles.zeroList, 'Value', 1);
0256
0257 %----- Update Model response -----
0258 function updateModelResp(handles)
0259
0260 gain = str2double(get(handles.gainEdit, 'String'));
0261 poles = getappdata(handles.main, 'poles');
0262 zeros = getappdata(handles.main, 'zeros');
0263
0264 pzm = pzmodel(gain, poles, zeros);
0265
0266 f1s = get(handles.f1Edit, 'String');
0267 f2s = get(handles.f2Edit, 'String');
0268 nfs = get(handles.nfEdit, 'String');
0269 fss = get(handles.fsEdit, 'String');
0270
0271 if isempty(f1s)
0272     f1 = getLowerFreq(pzm)/10;
0273 else
0274     f1 = str2double(f1s);
0275 end
0276 if isempty(f2s)
0277     f2 = getUpperFreq(pzm)*10;
0278 else
0279     f2 = str2double(f2s);
0280 end
0281 if isempty(nfs)
0282     nf = 1000;
0283 else
0284     nf = str2double(nfs);
0285 end
0286
0287 if ~isempty(pzm.poles) || ~isempty(pzm.zeros)
0288
0289     % make model response
0290     a = resp(pzm, f1, f2, nf);
0291     d = a.data;
0292     mx = d.x;
0293     my = d.y;
0294
0295     % make IIR response
0296     if isempty(fss)
0297         filt = miir(plist([param('pzmodel', pzm)]));
0298     else
0299         filt = miir(plist([param('pzmodel', pzm) param('fs', str2double(fss))]]));
0300     end
0301     setappdata(handles.main, 'filt', filt);
0302     set(handles.filterStrEdit, 'String', string(filt));
0303
0304     filtresp = resp(filt, plist(param('f', mx.')));
0305     d = filtresp.data;
0306     fx = d.x;
0307     fy = d.y;
0308
0309     axes(handles.magAxes) % Select the proper axes

```

```

0310 loglog(mx,abs(my), fx, abs(fy), 'r--')
0311 grid on;
0312 axis tight
0313 xlabel('');
0314 ylabel('Magnitude');
0315 legend('Pole/Zero model', 'IIR','Location','SouthWest')
0316
0317 axes(handles.phaseAxes) % Select the proper axes
0318 semilogx(mx, utils.math.phase(my), fx, utils.math.phase(fy), 'r--')
0319 grid on;
0320 axis tight
0321 xlabel('Frequency [Hz]');
0322 ylabel('Phase [deg]');
0323 else
0324 axes(handles.magAxes) % Select the proper axes
0325 cla
0326 axes(handles.phaseAxes) % Select the proper axes
0327 cla
0328 end
0329
0330
0331
0332
0333 function gainEdit_Callback(hObject, eventdata, handles)
0334 % hObject      handle to gainEdit (see GCBO)
0335 % eventdata    reserved - to be defined in a future version of MATLAB
0336 % handles      structure with handles and user data (see GUIDATA)
0337
0338 % Hints: get(hObject,'String') returns contents of gainEdit as text
0339 %          str2double(get(hObject,'String')) returns contents of gainEdit as a double
0340
0341 updateModelResp(handles)
0342
0343 % --- Executes during object creation, after setting all properties.
0344 function gainEdit_CreateFcn(hObject, eventdata, handles)
0345 % hObject      handle to gainEdit (see GCBO)
0346 % eventdata    reserved - to be defined in a future version of MATLAB
0347 % handles      empty - handles not created until after all CreateFcns called
0348
0349 % Hint: edit controls usually have a white background on Windows.
0350 %          See ISPC and COMPUTER.
0351 if ispc && isequal(get(hObject,'BackgroundColor'),
0352 get(0,'defaultUicontrolBackgroundColor'))
0353     set(hObject,'BackgroundColor','white');
0354 end
0355
0356
0357 function f1Edit_Callback(hObject, eventdata, handles)
0358 % hObject      handle to f1Edit (see GCBO)
0359 % eventdata    reserved - to be defined in a future version of MATLAB
0360 % handles      structure with handles and user data (see GUIDATA)
0361
0362 % Hints: get(hObject,'String') returns contents of f1Edit as text
0363 %          str2double(get(hObject,'String')) returns contents of f1Edit as a double
0364
0365
0366 % --- Executes during object creation, after setting all properties.
0367 function f1Edit_CreateFcn(hObject, eventdata, handles)
0368 % hObject      handle to f1Edit (see GCBO)
0369 % eventdata    reserved - to be defined in a future version of MATLAB
0370 % handles      empty - handles not created until after all CreateFcns called
0371
0372 % Hint: edit controls usually have a white background on Windows.
0373 %          See ISPC and COMPUTER.
0374 if ispc && isequal(get(hObject,'BackgroundColor'),
0375 get(0,'defaultUicontrolBackgroundColor'))
0376     set(hObject,'BackgroundColor','white');
0377 end
0378
0379
0380 function f2Edit_Callback(hObject, eventdata, handles)
0381 % hObject      handle to f2Edit (see GCBO)
0382 % eventdata    reserved - to be defined in a future version of MATLAB
0383 % handles      structure with handles and user data (see GUIDATA)
0384
0385 % Hints: get(hObject,'String') returns contents of f2Edit as text
0386 %          str2double(get(hObject,'String')) returns contents of f2Edit as a double
0387
0388
0389 % --- Executes during object creation, after setting all properties.
0390 function f2Edit_CreateFcn(hObject, eventdata, handles)
0391 % hObject      handle to f2Edit (see GCBO)
0392 % eventdata    reserved - to be defined in a future version of MATLAB
0393 % handles      empty - handles not created until after all CreateFcns called

```



```

0394
0395 % Hint: edit controls usually have a white background on Windows.
0396 % See ISPC and COMPUTER.
0397 if ispc && isequal(get(hObject,'BackgroundColor'),
0398 get(0,'defaultUiControlBackgroundColor'))
0399     set(hObject,'BackgroundColor','white');
0400 end
0401
0402
0403 function nfEdit_Callback(hObject, eventdata, handles)
0404 % hObject handle to nfEdit (see GCBO)
0405 % eventdata reserved - to be defined in a future version of MATLAB
0406 % handles structure with handles and user data (see GUIDATA)
0407
0408 % Hints: get(hObject,'String') returns contents of nfEdit as text
0409 % str2double(get(hObject,'String')) returns contents of nfEdit as a double
0410
0411
0412 % --- Executes during object creation, after setting all properties.
0413 function nfEdit_CreateFcn(hObject, eventdata, handles)
0414 % hObject handle to nfEdit (see GCBO)
0415 % eventdata reserved - to be defined in a future version of MATLAB
0416 % handles empty - handles not created until after all CreateFcns called
0417
0418 % Hint: edit controls usually have a white background on Windows.
0419 % See ISPC and COMPUTER.
0420 if ispc && isequal(get(hObject,'BackgroundColor'),
0421 get(0,'defaultUiControlBackgroundColor'))
0422     set(hObject,'BackgroundColor','white');
0423 end
0424
0425 % --- Executes on button press in replotBtn.
0426 function replotBtn_Callback(hObject, eventdata, handles)
0427 % hObject handle to replotBtn (see GCBO)
0428 % eventdata reserved - to be defined in a future version of MATLAB
0429 % handles structure with handles and user data (see GUIDATA)
0430
0431 updateModelResp(handles)
0432
0433
0434
0435 function fsEdit_Callback(hObject, eventdata, handles)
0436 % hObject handle to fsEdit (see GCBO)
0437 % eventdata reserved - to be defined in a future version of MATLAB
0438 % handles structure with handles and user data (see GUIDATA)
0439
0440 % Hints: get(hObject,'String') returns contents of fsEdit as text
0441 % str2double(get(hObject,'String')) returns contents of fsEdit as a double
0442 updateModelResp(handles)
0443
0444
0445 % --- Executes during object creation, after setting all properties.
0446 function fsEdit_CreateFcn(hObject, eventdata, handles)
0447 % hObject handle to fsEdit (see GCBO)
0448 % eventdata reserved - to be defined in a future version of MATLAB
0449 % handles empty - handles not created until after all CreateFcns called
0450
0451 % Hint: edit controls usually have a white background on Windows.
0452 % See ISPC and COMPUTER.
0453 if ispc && isequal(get(hObject,'BackgroundColor'),
0454 get(0,'defaultUiControlBackgroundColor'))
0455     set(hObject,'BackgroundColor','white');
0456 end
0457
0458 % --- Executes on button press in figureBtn.
0459 function figureBtn_Callback(hObject, eventdata, handles)
0460 % hObject handle to figureBtn (see GCBO)
0461 % eventdata reserved - to be defined in a future version of MATLAB
0462 % handles structure with handles and user data (see GUIDATA)
0463
0464 gain = str2double(get(handles.gainEdit, 'String'));
0465 poles = getappdata(handles.main, 'poles');
0466 zeros = getappdata(handles.main, 'zeros');
0467
0468 pzm = pzmodel(gain, poles, zeros);
0469
0470 f1s = get(handles.f1Edit, 'String');
0471 f2s = get(handles.f2Edit, 'String');
0472 nfs = get(handles.nfEdit, 'String');
0473 fss = get(handles.fsEdit, 'String');
0474
0475 if isempty(f1s)
0476     f1 = getLowerFreq(pzm)/10;

```

```

0477 else
0478     f1 = str2double(f1s);
0479 end
0480 if isempty(f2s)
0481     f2 = getupperFreq(pzm)*10;
0482 else
0483     f2 = str2double(f2s);
0484 end
0485 if isempty(nfs)
0486     nf = 1000;
0487 else
0488     nf = str2double(nfs);
0489 end
0490
0491 % make model response
0492 a = resp(pzm, f1, f2, nf);
0493 a.setName('PZmodel');
0494
0495 % make IIR response
0496 if isempty(fss)
0497     filt = miir(plist([param('pzmodel', pzm)]));
0498 else
0499     filt = miir(plist([param('pzmodel', pzm) param('fs', str2double(fss))]]));
0500 end
0501
0502 filtresp = resp(filt, plist(param('f', a.data.getX)));
0503 filtresp.setName('IIR filter');
0504
0505 iplot(a, filtresp)
0506
0507
0508
0509
0510 function filterStrEdit_Callback(hObject, eventdata, handles)
0511 % hObject    handle to filterStrEdit (see GCBO)
0512 % eventdata  reserved - to be defined in a future version of MATLAB
0513 % handles    structure with handles and user data (see GUIDATA)
0514
0515 % Hints: get(hObject,'String') returns contents of filterStrEdit as text
0516 %        str2double(get(hObject,'String')) returns contents of filterStrEdit as a double
0517
0518
0519 % --- Executes during object creation, after setting all properties.
0520 function filterStrEdit_CreateFcn(hObject, eventdata, handles)
0521 % hObject    handle to filterStrEdit (see GCBO)
0522 % eventdata  reserved - to be defined in a future version of MATLAB
0523 % handles    empty - handles not created until after all CreateFcns called
0524
0525 % Hint: edit controls usually have a white background on Windows.
0526 %        See ISPC and COMPUTER.
0527 if ispc && isequal(get(hObject,'BackgroundColor'),
0528 get(0,'defaultUicontrolBackgroundColor'))
0528     set(hObject,'BackgroundColor','white');
0529 end
0530
0531
0532 % --- Executes on button press in clearBtn.
0533 function clearBtn_Callback(hObject, eventdata, handles)
0534 % hObject    handle to clearBtn (see GCBO)
0535 % eventdata  reserved - to be defined in a future version of MATLAB
0536 % handles    structure with handles and user data (see GUIDATA)
0537
0538 setappdata(handles.main, 'poles', []);
0539 setappdata(handles.main, 'zeros', []);
0540
0541 % update list
0542 updateZerosList(handles);
0543 updatePolesList(handles);
0544
0545 % update model
0546 updateModelResp(handles);
0547
0548
0549 % --- Executes on button press in deletePolesBtn.
0550 function deletePolesBtn_Callback(hObject, eventdata, handles)
0551 % hObject    handle to deletePolesBtn (see GCBO)
0552 % eventdata  reserved - to be defined in a future version of MATLAB
0553 % handles    structure with handles and user data (see GUIDATA)
0554
0555 % get selection from list
0556 values = get(handles.poleList, 'Value');
0557
0558 % get poles
0559 poles = getappdata(handles.main, 'poles');
0560
0561 po = [];

```

```

0562 for j=1:length(poles)
0563     if j~=values
0564         po = [po poles(j)];
0565     end
0566 end
0567
0568 setappdata(handles.main, 'poles', po);
0569 updatePolesList(handles);
0570
0571 % update model
0572 updateModelResp(handles);
0573
0574
0575 % --- Executes on button press in deleteZerosBtn.
0576 function deleteZerosBtn_Callback(hObject, eventdata, handles)
0577 % hObject    handle to deleteZerosBtn (see GCBO)
0578 % eventdata  reserved - to be defined in a future version of MATLAB
0579 % handles    structure with handles and user data (see GUIDATA)
0580
0581 % get selection from list
0582 values = get(handles.zeroList, 'Value');
0583 % get zeros
0584 zeros = getappdata(handles.main, 'zeros');
0585 zo = [];
0586 for j=1:length(zeros)
0587     if j~=values
0588         zo = [zo zeros(j)];
0589     end
0590 end
0591
0592 setappdata(handles.main, 'zeros', zo);
0593 updateZerosList(handles);
0594
0595 % update model
0596 updateModelResp(handles);
0597
0598
0599 % --- Executes on button press in saveA0btn.
0600 function saveA0btn_Callback(hObject, eventdata, handles)
0601 % hObject    handle to saveA0btn (see GCBO)
0602 % eventdata  reserved - to be defined in a future version of MATLAB
0603 % handles    structure with handles and user data (see GUIDATA)
0604
0605 % get response
0606 gain = str2double(get(handles.gainEdit, 'String'));
0607 poles = getappdata(handles.main, 'poles');
0608 zeros = getappdata(handles.main, 'zeros');
0609 pzm = pzmodel(gain, poles, zeros);
0610
0611 f1s = get(handles.f1Edit, 'String');
0612 f2s = get(handles.f2Edit, 'String');
0613 nfs = get(handles.nfEdit, 'String');
0614 fss = get(handles.fsEdit, 'String');
0615
0616 if isempty(f1s)
0617     f1 = getlowerFreq(pzm)/10;
0618 else
0619     f1 = str2double(f1s);
0620 end
0621 if isempty(f2s)
0622     f2 = getupperFreq(pzm)*10;
0623 else
0624     f2 = str2double(f2s);
0625 end
0626 if isempty(nfs)
0627     nf = 1000;
0628 else
0629     nf = str2double(nfs);
0630 end
0631
0632 % make model response
0633 a = resp(pzm, f1, f2, nf);
0634
0635 % Get filename
0636 [filename, pathname] = uiputfile('*.xml', 'LTPDA XML file (*.xml)', 'Save as');
0637 if isequal(filename,0) || isequal(pathname,0)
0638     else
0639         fname = fullfile(pathname, filename);
0640         save(a, fname);
0641     end
0642

```


Contents

PURPOSE ^

%% path: m/gui/quicklook %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/gui/quicklook %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    <a href="matlab:help m/gui/quicklook/ltpdaquicklook">m/gui/quicklook/ltpdaquicklook</a> -
    allows the user to quicklook LTPDA objects.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpdaquicklook

PURPOSE ^

LTPDAQUICKLOOK allows the user to quicklook LTPDA objects.

SYNOPSIS ^

function varargout = ltpdaquicklook(varargin)

DESCRIPTION ^

```
LTPDAQUICKLOOK allows the user to quicklook LTPDA objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDAQUICKLOOK allows the user to quicklook LTPDA objects.
CALL:      ltpdaquicklook

VERSION:    $Id: ltpdaquicklook.m,v 1.4 2008/08/04 18:15:09 hewitson Exp $
HISTORY:    07-03-08 M Hewitson
            Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SUBFUNCTIONS ^

-  function call_histplot(varargin)
-  function call_iplot(varargin)
-  function refresh(varargin)
-  function ltpda_quicklook_close(varargin)
-  function objs = getWorkspaceObjs()
-  function objList(varargin)
-  function setWorkspaceObjsList(objs)

SOURCE CODE ^

```
0001 function varargout = ltpdaquicklook(varargin)
0002
0003 % LTPDAQUICKLOOK allows the user to quicklook LTPDA objects.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: LTPDAQUICKLOOK allows the user to quicklook LTPDA objects.
0008 %
0009 % CALL:      ltpdaquicklook
0010 %
0011 %
0012 % VERSION:    $Id: ltpdaquicklook.m,v 1.4 2008/08/04 18:15:09 hewitson Exp $
0013 %
0014 % HISTORY:    07-03-08 M Hewitson
0015 %            Creation
```

```

0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020 %% Check if I exist already
0021 hs = findall(0);
0022 found = -1;
0023 for j=1:length(hs)
0024     h = hs(j);
0025     if strcmp(get(h, 'Tag'), 'LTPDAquicklook')
0026         found = h;
0027     end
0028 end
0029 if found ~= -1
0030     figure(found);
0031     return
0032 end
0033
0034 % id = findobj('Tag', 'LTPDAquicklook');
0035 % if ~isempty(id)
0036 %     figure(id)
0037 %     return
0038 % end
0039
0040 %% Some initial setup
0041
0042 Gproperties.Gcol = [240 240 240]/255;
0043 Gproperties.Gwidth = 600;
0044 Gproperties.Gheight = 400;
0045 Gproperties.Gborder = 10;
0046 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0047
0048 Gproperties.Screen = get(0, 'screensize');
0049 Gproperties.Gposition = [150 ...
0050     100 ...
0051     Gproperties.Gwidth...
0052     Gproperties.Gheight];
0053
0054 % Initialize and hide the GUI as it is being constructed.
0055 mainfig = figure('Name', 'LTPDA Quicklook',...
0056     'NumberTitle','off',...
0057     'Visible','off',...
0058     'Position',Gproperties.Gposition,...
0059     'Color', Gproperties.Gcol,...
0060     'ToolBar','none',...
0061     'MenuBar','none',...
0062     'Resize','off',...
0063     'HandleVisibility','callback', ...
0064     'Tag', 'LTPDAquicklook');
0065
0066 % Set mainfig callbacks
0067 set(mainfig, 'CloseRequestFcn', {@ltpda_quicklook_close, mainfig});
0068
0069 % Set Application data
0070 setappdata(mainfig, 'Gproperties', Gproperties);
0071
0072
0073 %% GUI Parts
0074
0075 % Objects list
0076 objs = getworkspaceobjs();
0077 setappdata(mainfig, 'objs', objs);
0078
0079 lbh = uicontrol(mainfig, 'Style','listbox',...
0080     'String',{' '},...
0081     'Value',1,...
0082     'BackgroundColor','w',...
0083     'FontSize', fontsize,...
0084     'Max', 1000,...
0085     'Position',[10 90 150 300],...
0086     'Tag', 'LTPDA_quicklook_objlist');
0087
0088 % Set callback
0089 set(lbh, 'Callback', {@objList, mainfig});
0090
0091 setworkspaceobjsList(objs)
0092
0093 % Refresh button
0094 pbh = uicontrol(mainfig, 'Style','pushbutton',...
0095     'String','Refresh',...
0096     'Callback', {@refresh, mainfig}, ...
0097     'Position',[10 70 60 25]);
0098
0099
0100 % Object display
0101 sth = uicontrol(mainfig, 'Style','text',...

```

```

0102 'String',''....
0103 'BackgroundColor','w',...
0104 'ForegroundColor','b',...
0105 'HorizontalAlignment','left',...
0106 'Tag','LTPDA_quicklook_display',...
0107 'Position',[170 90 420 300]);
0108
0109 % Plot button
0110 pbh = uicontrol(mainfig,'Style','pushbutton',...
0111 'String','iplot',
0112 'Callback',{@call_iplot, mainfig}, ...
0113 'Position',[10 10 60 25]);
0114
0115 % history plot
0116 pbh = uicontrol(mainfig,'Style','pushbutton',...
0117 'String','plot history',...
0118 'Callback',{@call_histplot, mainfig}, ...
0119 'Position',[80 10 80 25]);
0120
0121
0122 %% Start the GUI
0123
0124 % Make the GUI visible.
0125 set(mainfig,'Visible','on')
0126
0127 %% Callbacks
0128
0129 %----- history plot
0130 function call_histplot(varargin)
0131
0132 mainfig = varargin{end};
0133 objs = getappdata(mainfig, 'objs');
0134
0135 % get selection
0136 olh = findobj(mainfig, 'Tag', 'LTPDA_quicklook_objlist');
0137 idx = get(olh, 'Value');
0138
0139 cmd = sprintf('obj = evalin(''base'', '%s');', objs(idx).name);
0140 eval(cmd);
0141
0142 plot(obj.hist);
0143
0144
0145 %----- iplot
0146 function call_iplot(varargin)
0147
0148 mainfig = varargin{end};
0149 objs = getappdata(mainfig, 'objs');
0150
0151 % get selection
0152 olh = findobj(mainfig, 'Tag', 'LTPDA_quicklook_objlist');
0153 idx = get(olh, 'Value');
0154
0155 cmd = sprintf('obj = evalin(''base'', '%s');', objs(idx).name);
0156 eval(cmd);
0157
0158 iplot(obj);
0159
0160 %----- refresh
0161 function refresh(varargin)
0162
0163 mainfig = varargin{end};
0164 objs = getWorkspaceObjs();
0165 setWorkspaceObjsList(objs)
0166 setappdata(mainfig, 'objs', objs);
0167
0168 %----- Close function
0169 function ltpda_quicklook_close(varargin)
0170 % Callback executed when the GUI is closed
0171
0172 disp('* Goodbye from the LTPDA Quicklook GUI *')
0173 delete(varargin{1})
0174
0175
0176 %----- Get a list of LTPDA objects in the MATLAB workspace
0177 function objs = getWorkspaceObjs()
0178
0179 % get base workspace variables
0180 ws_vars = evalin('base','whos');
0181
0182 objs = [];
0183 for j=1:length(ws_vars)
0184     cmd = sprintf('obj = evalin(''base'', '%s');', ws_vars(j).name);
0185     eval(cmd)
0186     if isa(obj, 'ltpda_uo')
0187         objs = [objs ws_vars(j)];

```

```
0188     end
0189 end
0190
0191
0192 %----- Callback for list selection
0193 function objList(varargin)
0194
0195     mainfig = varargin{end};
0196     objs = getappdata(mainfig, 'objs');
0197
0198     % get selection
0199     olh = findobj(mainfig, 'Tag', 'LTPDA_quicklook_objlist');
0200     idx = get(olh, 'Value');
0201
0202     cmd = sprintf('obj = evalin(''base'', '%s');', objs(idx).name);
0203     eval(cmd);
0204
0205     txt = display(obj);
0206
0207     dh = findobj(mainfig, 'Tag', 'LTPDA_quicklook_display');
0208     set(dh, 'String', txt);
0209
0210
0211 %----- Fill the workspace object list
0212 function setWorkspaceObjsList(objs)
0213
0214     id = findobj('Tag', 'LTPDA_quicklook_objlist');
0215
0216     objlist = [];
0217     for j=1:length(objs)
0218         obj = objs(j);
0219         str = sprintf('%s\t\t(%s)', obj.name, obj.class);
0220         objlist = [objlist cellstr(str)];
0221     end
0222
0223     set(id, 'Value', 1);
0224     set(id, 'String', objlist);
0225
0226
```

Contents

PURPOSE ^

%% path: m/gui/specwin_viewer
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%% path: m/gui/specwin_viewer %%%%%%%%%%

[m/gui/specwin_viewer/ltpda_specwin_viewer_build_window](matlab:help m/gui/specwin_viewer/ltpda_specwin_viewer_build_window)

- Function to build the window and display it.

[m/gui/specwin_viewer/ltpda_specwin_viewer_close](matlab:help m/gui/specwin_viewer/ltpda_specwin_viewer_close)

- Callback executed when the GUI is closed

[m/gui/specwin_viewer/ltpda_specwin_viewer_wintype](matlab:help m/gui/specwin_viewer/ltpda_specwin_viewer_wintype)

- Callback executed when the user selects a window

[m/gui/specwin_viewer/specwin_viewer](matlab:help m/gui/specwin_viewer/specwin_viewer)

- allows the user to explore spectral windows.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ltpda_specwin_viewer_build_window

PURPOSE ^

Function to build the window and display it.

SYNOPSIS ^

function ltpda_specwin_viewer_build_window(varargin)

DESCRIPTION ^

Function to build the window and display it.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin_viewer](#) SPECWIN_VIEWER allows the user to explore spectral windows.

SOURCE CODE ^

```
0001 function ltpda_specwin_viewer_build_window(varargin)
0002 % Function to build the window and display it.
0003
0004 plotType = varargin{end};
0005
0006 mainfig = findobj('Tag', 'LTPDAspecwin_viewer');
0007 ax      = getappdata(mainfig, 'axes');
0008 info    = findobj('Tag', 'LTPDA_specwin_viewer_wininfo');
0009
0010 % get window type
0011 winTxt = findobj('Tag', 'LTPDA_specwin_viewer_winType');
0012 wins   = get(winTxt, 'String');
0013 win    = wins{get(winTxt, 'Value')};
0014
0015 % Get window size
0016 sizeTxt = findobj('Tag', 'LTPDA_specwin_viewer_winSize');
0017 winlen  = str2num(get(sizeTxt, 'String'));
0018
0019 if strcmpi(win, 'Kaiser')
0020     % get psll
0021     psllTxt = findobj('Tag', 'LTPDA_specwin_viewer_PSLL');
0022     psll    = str2num(get(psllTxt, 'String'));
0023     w = specwin(win, winlen, psll);
0024     cstr = sprintf('specwin('%s'', %d, %f)', win, winlen, psll);
0025 else
0026     w = specwin(win, winlen);
0027     cstr = sprintf('specwin('%s'', %d)', win, winlen);
0028 end
0029
0030 constTxt = findobj(mainfig, 'Tag', 'LTPDA_specwin_viewer_cstr');
0031 set(constTxt, 'String', cstr);
0032
0033
0034 titleStr = '';
0035 legendStr = '';
0036
0037 switch plotType
0038 case 'Time-domain'
0039     hl = plot(ax, 1:length(w.win), w.win);
0040     xlabel('sample');
0041     ylabel('amplitude');
0042 case 'Freq-domain'
0043     % Freq response
```

```

0044     f = linspace(-30,30,1000);
0045     r = zeros(size(f));
0046     N = length(w.win);
0047
0048     % do dft
0049     for j=1:length(f)
0050         k = [0:N-1].';
0051         r(j) = (w.win * exp(-2*pi*1i*f(j)*k/N) ) ./ w.ws;
0052     end
0053
0054     % convert to dB
0055     dbr = 20*log10(abs(r));    hl = plot(ax, f, dbr);
0056     xlabel('bin');
0057     ylabel('amplitude [dB]');
0058 end
0059
0060 col = [0.8 0.1 0.1];
0061 set(hl, 'Color', col);
0062 titleStr = [titleStr, utils.plottools.label(w.type), ', '];
0063 titleStr = titleStr(1:end-2);
0064 title(sprintf('Window: %s', titleStr));
0065 axis tight
0066 grid on
0067
0068
0069 % Info string
0070 lstr = [sprintf('alpha = %g\n', w.alpha)...
0071        sprintf('psll = %g\n', w.psll)...
0072        sprintf('rov = %g\n', w.rov)...
0073        sprintf('nenbw = %g\n', w.nenbw)...
0074        sprintf('w3db = %g\n', w.w3db)...
0075        sprintf('flatness = %g\n', w.flatness)];
0076 legendStr = [legendStr cellstr(lstr)];
0077 set(info, 'String', legendStr);
0078
0079 % Constructor
0080
0081
0082
0083
0084

```

ltpda_specwin_viewer_close

PURPOSE ^

Callback executed when the GUI is closed

SYNOPSIS ^

function ltpda_specwin_viewer_close(varargin)

DESCRIPTION ^

Callback executed when the GUI is closed

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 function ltpda_specwin_viewer_close(varargin)
0002 % Callback executed when the GUI is closed
0003
0004 disp('* Goodbye from the LTPDA Spectral Window Viewer *')
0005 delete(varargin{1})
0006
0007 end
0008
```


ltpda_specwin_viewer_wintype

PURPOSE ^

Callback executed when the user selects a window

SYNOPSIS ^

function ltpda_specwin_viewer_wintype(varargin)

DESCRIPTION ^

Callback executed when the user selects a window

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

 [specwin_viewer](#) SPECWIN_VIEWER allows the user to explore spectral windows.

SOURCE CODE ^

```
0001 function ltpda_specwin_viewer_wintype(varargin)
0002 % Callback executed when the user selects a window
0003
0004
0005 etxt = findobj('Tag', 'LTPDA_specwin_viewer_PSL');
0006 wsel = findobj('Tag', 'LTPDA_specwin_viewer_winType');
0007
0008 idx = get(wsel, 'Value');
0009 wins = get(wsel, 'String');
0010 win = wins{idx};
0011
0012
0013 if strcmpi(win, 'Kaiser')
0014     set(etxt, 'Enable', 'on');
0015 else
0016     set(etxt, 'Enable', 'off');
0017 end
0018
0019
0020
0021 end
0022
```


specwin_viewer

PURPOSE ^

SPECWIN_VIEWER allows the user to explore spectral windows.

SYNOPSIS ^

function varargout = specwin_viewer(varargin)

DESCRIPTION ^

```
SPECWIN_VIEWER allows the user to explore spectral windows.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: SPECWIN_VIEWER allows the user to explore spectral windows.
CALL:      specwin_viewer

VERSION:    $Id: specwin_viewer.m,v 1.4 2008/08/04 18:23:04 hewitson Exp $
HISTORY:    07-03-08 M Hewitson
            Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

-  [ltpda_specwin_viewer_build_window](#) Function to build the window and display it.
-  [ltpda_specwin_viewer_wintype](#) Callback executed when the user selects a window

This function is called by:

SUBFUNCTIONS ^

-  [function buildWindowSelectionMenu\(mainfig\)](#)

SOURCE CODE ^

```
0001 function varargout = specwin_viewer(varargin)
0002
0003 % SPECWIN_VIEWER allows the user to explore spectral windows.
0004 %
0005 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0006 %
0007 % DESCRIPTION: SPECWIN_VIEWER allows the user to explore spectral windows.
0008 %
0009 % CALL:      specwin_viewer
0010 %
0011 %
0012 % VERSION:    $Id: specwin_viewer.m,v 1.4 2008/08/04 18:23:04 hewitson Exp $
0013 %
0014 % HISTORY:    07-03-08 M Hewitson
0015 %            Creation
0016 %
0017 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0018 %
0019 %% Check if I exist already
0020 id = findobj('Tag', 'LTPDAspecwin viewer');
```

```

0022 if ~isempty(id)
0023     figure(id)
0024     return
0025 end
0026
0027 %% Some initial setup
0028
0029 Gproperties.Gcol      = [240 240 240]/255;
0030 Gproperties.Gwidth    = 800;
0031 Gproperties.Gheight   = 400;
0032 Gproperties.Gborder   = 10;
0033 fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
0034
0035 Gproperties.Screen    = get(0, 'screensize');
0036 Gproperties.Gposition = [150 ...
0037     150 ...
0038     Gproperties.Gwidth...
0039     Gproperties.Gheight];
0040
0041 % Initialize and hide the GUI as it is being constructed.
0042 mainfig = figure('Name', 'LTPDA Spectral Window Viewer',...
0043     'NumberTitle', 'off',...
0044     'Visible', 'off',...
0045     'Position', Gproperties.Gposition,...
0046     'Color', Gproperties.Gcol,...
0047     'ToolBar', 'none',...
0048     'NextPlot', 'new',...
0049     'MenuBar', 'none',...
0050     'Resize', 'off',...
0051     'Tag', 'LTPDAspecwin_viewer');
0052
0053 % Set mainfig callbacks
0054 set(mainfig, 'CloseRequestFcn', {'ltpda_specwin_viewer_close', mainfig});
0055
0056 % Set Application data
0057 setappdata(mainfig, 'Gproperties', Gproperties);
0058
0059 %% Window selection dialog
0060
0061 %----- Window selection box
0062 buildWindowSelectionMenu(mainfig);
0063
0064 %----- Window size
0065
0066 sth = uicontrol(mainfig, 'Style', 'text',...
0067     'String', 'Window Size',...
0068     'BackgroundColor', Gprops.Gcol, ...
0069     'Position', [10 Gprops.Gheight-70 80 25]);
0070
0071 eth = uicontrol(mainfig, 'Style', 'edit',...
0072     'String', '100',...
0073     'BackgroundColor', 'w',...
0074     'Tag', 'LTPDA_specwin_viewer_WinSize',...
0075     'Position', [100 Gprops.Gheight-70 120 25]);
0076
0077 %----- PSLL
0078
0079 sth = uicontrol(mainfig, 'Style', 'text',...
0080     'String', 'Window PSLL',...
0081     'BackgroundColor', Gprops.Gcol, ...
0082     'Position', [10 Gprops.Gheight-100 80 25]);
0083
0084 eth = uicontrol(mainfig, 'Style', 'edit',...
0085     'String', '150',...
0086     'BackgroundColor', 'w',...
0087     'Tag', 'LTPDA_specwin_viewer_PSLL',...
0088     'Position', [100 Gprops.Gheight-100 120 25]);
0089
0090 %----- Build Time-domain Button
0091 pbh = uicontrol(mainfig, 'Style', 'pushbutton',...
0092     'String', 'Plot Time-domain',...
0093     'Callback', {'ltpda_specwin_viewer_build_window', 'Time-domain'},...
0094     'Position', [10 Gprops.Gheight-140 100 25]);
0095
0096 %----- Build Freq-domain Button
0097 pbh = uicontrol(mainfig, 'Style', 'pushbutton',...
0098     'String', 'Plot Freq-domain',...
0099     'Callback', {'ltpda_specwin_viewer_build_window', 'Freq-domain'},...
0100     'Position', [120 Gprops.Gheight-140 100 25]);
0101
0102 %----- Axes
0103
0104 M = 0.01;
0105 ax = axes('Position', [0.5+M 0.2 0.4 0.7], ...
0106     'Parent', mainfig, ...

```



```

0108         'Tag', 'LTPDA_specwin_viewer_axes', ...
0109         'Visible', 'on', ...
0110         'Box', 'on');
0111
0112 setappdata(mainfig, 'axes', ax);
0113
0114 sth = uicontrol(mainfig, 'Style', 'text', ...
0115     'String', 'None', ...
0116     'BackgroundColor', Gprops.Gcol, ...
0117     'Tag', 'LTPDA_specwin_viewer_wininfo', ...
0118     'Position', [10 10 300 Gproperties.Gheight/2 - 20], ...
0119     'FontSize', 14, 'ForegroundColor', 'b');
0120
0121 % Constructor text field
0122 sth = uicontrol(mainfig, 'Style', 'edit', ...
0123     'String', '', ...
0124     'BackgroundColor', 'w', ...
0125     'Tag', 'LTPDA_specwin_viewer_cstr', ...
0126     'Position', [10 10 300 25]);
0127
0128
0129 % call callback for window type
0130 ltpda_specwin_viewer_wintype(mainfig);
0131
0132 % draw window
0133 ltpda_specwin_viewer_build_window('Time-domain');
0134
0135
0136
0137
0138 %% Start the GUI
0139
0140 % Make the GUI visible.
0141 set(mainfig, 'Visible', 'on')
0142
0143
0144 %% Sub-functions
0145
0146 %---- Window selection box
0147 function buildWindowSelectionMenu(mainfig)
0148
0149     Gprops = getappdata(mainfig, 'Gproperties');
0150
0151     % get window list
0152     wins = specwin.getTypes;
0153
0154     % text field
0155     sth = uicontrol(mainfig, 'Style', 'text', ...
0156         'String', 'Window Type', ...
0157         'BackgroundColor', Gprops.Gcol, ...
0158         'Position', [10 Gprops.Gheight-40 80 25]);
0159
0160     % pop-up dialog
0161     pmh = uicontrol(mainfig, 'Style', 'popupmenu', ...
0162         'String', [{ 'Kaiser' } wins], ...
0163         'Value', 1, ...
0164         'BackgroundColor', Gprops.Gcol, ...
0165         'Tag', 'LTPDA_specwin_viewer_winType', ...
0166         'Callback', {'ltpda_specwin_viewer_wintype', mainfig}, ...
0167         'Position', [100 Gprops.Gheight-40 120 25]);
0168
0169 end
0170
0171 end
0172
0173
0174

```

Contents

PURPOSE ^

%% path: m/helper %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%% path: m/helper %%%
 [m/helper/mc](matlab:help m/helper/mc) - A function to properly clear MATLAB
memory for LTPDA.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

mc

PURPOSE ^

A function to properly clear MATLAB memory for LTPDA.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

A function to properly clear MATLAB memory for LTPDA.
M Hewitson 06-08-08
\$Id: mc.m,v 1.2 2008/08/06 18:25:47 hewitson Exp \$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % A function to properly clear MATLAB memory for LTPDA.
0002 %
0003 % M Hewitson 06-08-08
0004 %
0005 % $Id: mc.m,v 1.2 2008/08/06 18:25:47 hewitson Exp $
0006 %
0007 %
0008 % Remove this object from memory so we can clear classes
0009 setappdata(0, 'ltpda_default_spectral_window', '');
0010
0011 % clear
0012 clear
0013 clear classes
0014
0015 % set this global propery again
0016 setappdata(0, 'ltpda_default_spectral_window', specwin('BH92', 10));
0017
```


Contents

PURPOSE ^

%% path: m/math %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

%% path: m/math %%%

 [m/math/ltpda_lincom](matlab:help m/math/ltpda_lincom) - make the linear
 combination of the input analysis objects.

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Constructor examples of the PROVENANCE class

[Construct empty PROVENANCE object](#)

[Construct a PROVENANCE object by loading the object from a file](#)

[Construct a PROVENANCE object with a 'creator' name](#)

Poles are specified by in the LTPDA Toolbox by a frequency, f, and (optionally) a quality factor, Q.

Construct empty PROVENANCE object

The following example creates an empty provenance object

```
p = provenance()
----- provenance 01 -----
      creator:  diepholz
      created:  2008-03-30 17:35:33.943
         ip:   127.0.1.1
      hostname:  HWS169
         os:    GLNXA64
MATLAB ver:    7.5 (R2007b)
Sig Proc ver:  6.8 (R2007b)
      LTPDA ver: 9.900000e-01 (R2007b)
-----
```

Construct a PROVENANCE object by loading the object from a file

The following example creates a new provenance object by loading the provenance object from disk.

```
p = provenance('provenance.mat')
p = provenance('provenance.xml')
```

Construct a PROVENANCE object with a 'creator' name

If a 'creator' name is specified then creates the constructor a PROVENANCE object with this name as the creator.

```
p = provenance('other user')
----- provenance 01 -----
      creator:  other user
      created:  2008-03-30 17:36:52.369
         ip:   127.0.1.1
      hostname:  HWS169
         os:    GLNXA64
MATLAB ver:    7.5 (R2007b)
Sig Proc ver:  6.8 (R2007b)
```


LTPDA ver: 9.900000e-01 (R2007b)

◀ Constructor examples of the POLE class

Constructor examples of the PZMODEL class ▶

©LTP Team

Constructor examples of the XYZDATA class

[Construct empty XYZDATA object](#)

[Construct a XYZDATA object by loading the object from a file](#)

[Construct a XYZDATA object from z-data](#)

[Construct a XYZDATA object from a x-data, y-data and z-data](#)

Construct empty XYZDATA object

The following example creates an empty xyzdata object

```
xyz = xyzdata()
----- xyzdata 01 -----

name:    None
  x:    [0 0], double
  y:    [0 0], double
  z:    [0 0], double
xunits:
yunits:
zunits:
-----
```

Construct a XYZDATA object by loading the object from a file

The following example creates a new xyzdata object by loading the xyzdata object from disk.

```
xyz = xyzdata('xyz.mat')
xyz = xyzdata('xyz.xml')
```

Construct a XYZDATA object from z-data

Creates an xyz object with the given z-data.

```
z = randn(1000);
xyz = xyzdata(z)
----- xyzdata 01 -----

name:    None
  x:    [1000 1], double
  y:    [1000 1], double
  z:    [1000 1000], double
xunits:
yunits:
zunits:
-----
```

Construct a XYZDATA object from a x-data, y-data and z-data

creates an xy-data object with the given (x,y)-data.

```
x = randn(1000,1);
y = randn(1000,1);
z = x*y';

xy = xyzdata(x,y,z)
----- xyzdata 01 -----

name:    None
  x:    [1000 1], double
  y:    [1000 1], double
  z:    [1000 1000], double
xunits:
yunits:
zunits:
-----
```

Index for m/ssm_models

Matlab files in this directory:

 Contents	%% path: m/ssm_models %%
 ssm_model_delay	SSM_MODEL_DELAY defines a delay block, discrete
 ssm_model_standard_system_noparams	defines a simple 2nd order stable system with a control and a noise input
 ssm_model_standard_system_params	SSM_MODEL_STANDARD_SYSTEM_PARAMS

Subsequent directories:

 [CVS](#)

Contents

PURPOSE ^

%% path: m/sigproc/frequency_domain
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/sigproc/frequency_domain %%%%%%%%%%%%%%  
  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_cohere">m/sigproc/frequency_domain/ltpda_cohere</a> -  
makes coherence estimates of the time-series objects  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_cpsd">m/sigproc/frequency_domain/ltpda_cpsd</a> -  
makes cross-spectral density estimates of the time-series objects.  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_lcohere">m/sigproc/frequency_domain/ltpda_lcohere</a> -  
implement weighted coherence estimation computed on a log frequency axis.  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_lcpsd">m/sigproc/frequency_domain/ltpda_lcpsd</a> -  
implement weighted cross-power estimation computed on a log frequency axis.  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_linedetect">m/sigproc/frequency_domain/ltpda_linedetect</a> -  
find spectral lines in the ao/fsdata objects.  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_lpsd">m/sigproc/frequency_domain/ltpda_lpsd</a> -  
implement LPSD algorithm for analysis objects.  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_ltfe">m/sigproc/frequency_domain/ltpda_ltfe</a> -  
implement transfer-function estimation computed on a log frequency axis.  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_nfest">m/sigproc/frequency_domain/ltpda_nfest</a> -  
Estimate the noise floor of the given frequency-series A0.  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_pwelch">m/sigproc/frequency_domain/ltpda_pwelch</a> -  
makes power spectral density estimates of the time-series objects  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_rms">m/sigproc/frequency_domain/ltpda_rms</a> -  
Calculate RMS deviation from spectrum  
    <a href="matlab:help  
m/sigproc/frequency_domain/ltpda_tfe">m/sigproc/frequency_domain/ltpda_tfe</a> -  
makes transfer function estimates of the time-series objects.  
    <a href="matlab:help  
m/sigproc/frequency_domain/phasetrack">m/sigproc/frequency_domain/phasetrack</a> - %  
Check if this is a call for parameters, the CVS version string
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% path: m/sigproc/time_domain
%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/sigproc/time_domain %%%%%%%%%%  
    <a href="matlab:help  
m/sigproc/time_domain/ltpda_gapfilling">m/sigproc/time_domain/ltpda_gapfilling</a>    - fills  
possible gaps in data.  
    <a href="matlab:help  
m/sigproc/time_domain/ltpda_polydetrend">m/sigproc/time_domain/ltpda_polydetrend</a>    -  
detrends the input analysis object using a polynomial of degree N.  
    <a href="matlab:help  
m/sigproc/time_domain/ltpda_spikecleaning">m/sigproc/time_domain/ltpda_spikecleaning</a> - %  
LTPDA_spikecleaning detects and corrects possible spikes in analysis objects  
    <a href="matlab:help  
m/sigproc/time_domain/ltpda_xcorr">m/sigproc/time_domain/ltpda_xcorr</a>            - makes  
cross-correlation estimates of the time-series
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

Contents

PURPOSE ^

%% path: m/ssm_models %%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% path: m/ssm_models %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
- <a href="matlab:help m/ssm_models/ssm_model_delay">m/ssm_models/ssm_model_delay</a>
  defines a delay block, discrete
- <a href="matlab:help
m/ssm_models/ssm_model_standard_system_noparams">m/ssm_models/ssm_model_standard_system_noparams</a>
  % defines a simple 2nd order stable system with a control and a noise input
- <a href="matlab:help
m/ssm_models/ssm_model_standard_system_params">m/ssm_models/ssm_model_standard_system_params</a>
-
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

ssm_model_delay

PURPOSE ^

SSM_MODEL_DELAY defines a delay block, discrete

SYNOPSIS ^

function sys = ssm_model_delay(sys)

DESCRIPTION ^

SSM_MODEL_DELAY defines a delay block, discrete
VERSION: \$Id:\$

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % SSM_MODEL_DELAY defines a delay block, discrete
0002 %
0003 % VERSION: $Id:$
0004 %
0005 function sys = ssm_model_delay(sys)
0006
0007 % parameters in subsystem
0008 sys.paramnames ={'Tau'};
0009 sys.paramvalues = [0];
0010 sys.paramsigmas = [0.05];
0011
0012 %% declaring symbolic parameters
0013 for i_params=1:length(sys.paramnames)
0014     cmd = [sys.paramnames{i_params}, '=sym(''', sys.paramnames{i_params}, ''');'];
0015     eval(cmd);
0016 end
0017
0018 %%%%%%%%%%%%%%%%%%%%%%%%%%% START BUILDING PARAMETRIC SUBSYSTEM
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%
0020 Ts = 0.1;
0021 Tt = 0.3;
0022
0023 A1 = [...
0024     0 1 0 0 0 ;...
0025     0 0 1 0 0 ;...
0026     0 0 0 1 0 ;...
0027     0 0 0 0 1 ;...
0028     0 0 0 0 0];
0029 B1 = [0; 0; 0; 0; 1];
0030 C1 = [Tau 1-Tau 0 0 0];
0031 D1 = 0;
0032 A = blkdiag(A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1);
0033 B = blkdiag(B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1);
0034 C = blkdiag(C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1);
0035 D = blkdiag(D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1);
0036
0037 %%%%%%%%%%%%%%%%%%%%%%%%%%% END BUILDING PARAMETRIC SUBSYSTEM
0038 %%%%%%%%%%%%%%%%%%%%%%%%%%%
0039 sys.name = 'Delay';
0040
0041 sys.amats = {A};
0042 sys.bmats = {B};
```

```

0043 sys.cmats      = {C};
0044 sys.dmats      = {D};
0045 sys.timestep   = 0.1;
0046
0047 sys.ssnames     = {'De lay'};
0048
0049 sys.inputnames  = {'DFACS_command'};
0050 sys.inputvarnames = {'fx_mps_c' 'fy_mps_c' 'fz_mps_c' 'tx_mps_c' 'ty_mps_c' 'tz_mps_c'
...
0051 'fx1_TMAct_c' 'fy1_TMAct_c' 'fz1_TMAct_c' 'tx1_TMAct_c' 'ty1_TMAct_c' 'tz1_TMAct_c'
...
0052 'fx2_TMAct_c' 'fy2_TMAct_c' 'fz2_TMAct_c' 'tx2_TMAct_c' 'ty2_TMAct_c'
'tz2_TMAct_c'};
0053
0054 sys.outputnames = {...
0055 'fx_mps_c_delayed' 'fy_mps_c_delayed' 'fz_mps_c_delayed' 'tx_mps_c_delayed'
'ty_mps_c_delayed' 'tz_mps_c_delayed' ...
0056 'fx1_TMAct_c_delayed' 'fy1_TMAct_c_delayed' 'fz1_TMAct_c_delayed'
'tx1_TMAct_c_delayed' 'ty1_TMAct_c_delayed' 'tz1_TMAct_c_delayed' ...
0057 'fx2_TMAct_c_delayed' 'fy2_TMAct_c_delayed' 'fz2_TMAct_c_delayed'
'tx2_TMAct_c_delayed' 'ty2_TMAct_c_delayed' 'tz2_TMAct_c_delayed' ...
0058 };
0059 sys.outputvarnames = {'DFACS_command_delayed'};
0060
0061 end

```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

[Home](#) > [m](#) > [ssm_models](#) > ssm_model_standard_system_noparams.m

ssm_model_standard_system_noparams

PURPOSE [^]

defines a simple 2nd order stable system with a control and a noise input

SYNOPSIS [^]

```
function sys = ssm_model_standard_system_noparams(sys)
```

DESCRIPTION [^]

defines a simple 2nd order stable system with a control and a noise input

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SOURCE CODE [^]

```
0001
0002
0003 function sys = ssm_model_standard_system_noparams(sys)
0004 % defines a simple 2nd order stable system with a control and a noise input
0005
0006 sys.name = 'standard test system';
0007
0008 sys.amats = {[0 1 ; -0.04 -0.1]};
0009 sys.cmats = {[1 0]};
0010 sys.bmats = {[0;1] [0 0; 1 0]};
0011 sys.dmats = {1 [0 1]};
0012 sys.timestep = 0;
0013
0014 sys.ssnames = {'standard test system'};
0015 sys.ssvarnames = {'x' 'xdot'};
0016
0017 sys.inputnames = {'U' 'N'};
0018 sys.inputvarnames = {'Fu' 'Fn' 'On'};
0019
0020 sys.paramnames = {};
0021 sys.paramvalues = [];
0022 sys.paramsigmas = [];
0023
0024 sys.outputnames = {'Y'};
0025 sys.outputvarnames = {'y'};
0026 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ssm_model_standard_system_params

PURPOSE ^

SSM_MODEL_STANDARD_SYSTEM_PARAMS

SYNOPSIS ^

function sys = ssm_model_standard_system_params(sys)

DESCRIPTION ^

SSM_MODEL_STANDARD_SYSTEM_PARAMS

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % SSM_MODEL_STANDARD_SYSTEM_PARAMS
0002
0003 function sys = ssm_model_standard_system_params(sys)
0004     sys.name = 'standard test system';
0005
0006     sys.paramnames = {'w' 'c' 'c1' 'c2' 'b' 'd'};
0007     sys.paramvalues = [0.2 0.5 0 0 1 1];
0008     sys.paramsigmas = [0.2 0.1 1 1 1 1];
0009
0010 % declaring symbolic parameters
0011 for i_params=1:length(sys.paramnames)
0012     cmd = [sys.paramnames{i_params}, '=sym(''', sys.paramnames{i_params}, ''');'];
0013     eval(cmd);
0014 end
0015
0016 sys.amats = {[0 1 ; -w*w -w*c]};
0017 sys.cmats = {[1+c1 c2]};
0018 sys.bmats = {[0;b] [0 0; 1 0]};
0019 sys.d mats = {[d] [0 1]};
0020 sys.timestep = 0;
0021
0022 sys.ssnames = {'standard test system'};
0023 sys.ssvarnames = {'x' 'xdot'};
0024
0025 sys.inputnames = {'U' 'N'};
0026 sys.inputvarnames = {'Fu' 'Fn' 'On'};
0027
0028 sys.outputnames = {'Y'};
0029 sys.outputvarnames = {'y'};
0030 end
```


[Home](#) > [m](#) > [sigproc](#) > [frequency_domain](#) > [phasetrack.m](#)

phasetrack

PURPOSE [^]

SYNOPSIS [^]

```
function varargout = phasetrack(varargin)
```

DESCRIPTION [^]

CROSS-REFERENCE INFORMATION [^]

This function calls:

This function is called by:

SUBFUNCTIONS [^]

 [function plo = getDefaultPL\(\)](#)

SOURCE CODE [^]

```
0001 function varargout = phasetrack(varargin)
0002
0003 ALGONAME = mfilename;
0004 VERSION = '$Id: phasetrack.m,v 1.1 2008/05/21 09:53:13 anneke Exp $';
0005 CATEGORY = 'Signal Processing';
0006
0007 %% Check if this is a call for parameters, the CVS version string
0008 % or the function category
0009 if nargin == 1 && ischar(varargin{1})
0010     in = char(varargin{1});
0011     if strcmp(in, 'Params')
0012         varargout{1} = getDefaultPL();
0013         return
0014     elseif strcmp(in, 'Version')
0015         varargout{1} = VERSION;
0016         return
0017     elseif strcmp(in, 'Category')
0018         varargout{1} = CATEGORY;
0019         return
0020     end
0021 end
0022
0023
0024 invars = {};
0025 for j=1:nargin
0026     invars = [invars cellstr(inputname(j))];
0027 end
0028
0029
0030
0031 as = [];
0032 pl = [];
0033 for j=1:nargin
0034     a = varargin{j};
0035     if isa(a, 'plist')
0036         pl = a;
0037     end
0038     if isa(a, 'ao')
0039         for k=1:length(a)
0040             ak = a(k);
0041             d = ak.data;
0042             if isa(d, 'tsdata')
0043                 as = [as ak];
0044             end
0045         end
0046     end
0047 end
```

```

0044         else
0045             warning('### works only for time series');
0046         end
0047     end
0048 end
0049 end
0050 na = length(as);
0051 % unpack parameter list
0052 plo = plist();
0053
0054 % Initialise output
0055 bo = [];
0056
0057 for i=1:na
0058
0059     % get data out
0060     a = as(i);
0061     d = a.data;
0062     dinfo = whos('d');
0063     if ~isa(d, 'tsdata')
0064         error('### I only work with time-series at the moment.');
```

```

0065     end
0066     if ~isreal(d.y)
0067         error('### I only work with real time-series at the moment.');
```

```

0068     end
0069     add = 0;
0070     ydata = a.data.y;
0071     y = ydata;
0072
0073     for i = 2:length(ydata)
0074
0075         diff = ydata(i)-ydata(i-1);
0076         if diff > pi/2
0077             add = add-pi;
0078         elseif diff < -pi/2
0079             add = add+pi;
0080         end
0081         y(i) = ydata(i) + add;
0082     end
0083     % Make output analysis object
0084     nameStr = sprintf('phasetrack(%s)', d.name);
0085
0086     % create new output data
0087     data = tsdata(y, d.fs);
0088     data = set(data, 'name', nameStr);
0089     data = set(data, 'xunits', d.xunits);
0090     data = set(data, 'yunits', d.yunits);
0091
0092     % create new output history
0093     h = history(ALGONAME, VERSION, pl, a.hist);
0094     h = set(h, 'invars', invars);
0095
0096     % make output analysis object
0097     b = ao(data, h);
0098
0099     % set name
0100     % name for this object
0101     if isempty(invars{j})
0102         n1 = a.name;
0103     else
0104         n1 = invars{j};
0105     end
0106
0107     nameStr = sprintf('phasetrack(%s)', n1);
0108     b = setnh(b, 'name', nameStr);
0109
0110     % Add to output array
0111     bo = [bo b];
0112
0113 end
0114 varargout{1} = bo;
0115
0116 %-----
0117 % Get default params
0118 function plo = getDefaultPL()
0119
0120 disp('* creating default plist...');
0121 plo = plist();
0122 disp('* done.');
```

```

0123
0124
0125
0126 % END
0127
0128

```


ltpda_linedetect

PURPOSE ^

LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.

SYNOPSIS ^

function varargout = ltpda_linedetect(varargin)

DESCRIPTION ^

```
LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DESCRIPTION: LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.

CALL:      b = ltpda_linedetect(a, pl)

PARAMETERS: N      - max number of lines to return [default: 10]
             fsearch - frequeuncy search interval [default: all]
             thresh  - a threshold to test normalised amplitude spectrum against
                       [default: 2]
             bw      - bandwidth over which to compute median [default: 20 samples]
             hc      - percent of outliers to exclude from median estimation (0-1)
                       [default: 0.8]

VERSION:    $Id: ltpda_linedetect.m,v 1.10 2008/08/10 20:16:49 hewitson Exp $

HISTORY:    14-05-2007 M Hewitson
             Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: LTPDA_LINEDETECT find spectral lines in the ao/fsdata objects.
0005 %
0006 % CALL:      b = ltpda_linedetect(a, pl)
0007 %
0008 % PARAMETERS: N      - max number of lines to return [default: 10]
0009 %             fsearch - frequeuncy search interval [default: all]
0010 %             thresh  - a threshold to test normalised amplitude spectrum against
0011 %                       [default: 2]
0012 %             bw      - bandwidth over which to compute median [default: 20 samples]
0013 %             hc      - percent of outliers to exclude from median estimation (0-1)
0014 %                       [default: 0.8]
0015 %
0016 % VERSION:    $Id: ltpda_linedetect.m,v 1.10 2008/08/10 20:16:49 hewitson Exp $
0017 %
0018 % HISTORY:    14-05-2007 M Hewitson
0019 %             Creation
0020 %
0021 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0022 %
0023 function varargout = ltpda_linedetect(varargin)
0024 %
0025 warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/linedetect now.');
```

```
0026
0027 % Call ao/cohere
0028 bs = linedetect(varargin{:});
0029
0030 % set output
0031 varargout{1} = bs;
0032
0033
0034 end
```

ltpda_nfest

PURPOSE ^

LTPDA_NFEST Estimate the noise floor of the given frequency-series AO.

SYNOPSIS ^

function varargout = ltpda_nfest(varargin)

DESCRIPTION ^

```
LTPDA_NFEST Estimate the noise floor of the given frequency-series AO.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_NFEST Estimate the noise floor of the given
              frequency-series AO. This essentially implements a running
              median filter with some outlier selection.
CALL:        b = ltpda_nfest(a, pl)
PARAMETERS:
    bw - the bandwidth of the noise floor estimate [default: 20 samples]
    hc - a cutoff to throw away outliers (0-1) [default: 0.8]
VERSION: $Id: ltpda_nfest.m,v 1.9 2008/08/10 20:16:49 hewitson Exp $
HISTORY: 14-05-07 M Hewitson
          Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_NFEST Estimate the noise floor of the given frequency-series AO.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_NFEST Estimate the noise floor of the given
0006 %              frequency-series AO. This essentially implements a running
0007 %              median filter with some outlier selection.
0008 %
0009 % CALL:        b = ltpda_nfest(a, pl)
0010 %
0011 % PARAMETERS:
0012 %     bw - the bandwidth of the noise floor estimate [default: 20 samples]
0013 %     hc - a cutoff to throw away outliers (0-1) [default: 0.8]
0014 %
0015 % VERSION: $Id: ltpda_nfest.m,v 1.9 2008/08/10 20:16:49 hewitson Exp $
0016 %
0017 % HISTORY: 14-05-07 M Hewitson
0018 %          Creation
0019 %
0020 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0021 %
0022 function varargout = ltpda_nfest(varargin)
0023 %
0024     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/smoother now.');
```

```
0027     bs = smoother(varargin{:});
0028
0029     % set output
0030     varargout{1} = bs;
0031
0032 end
0033
0034
```


ltpda_polydetrend

PURPOSE ^

LTPDA_POLYDETREND detrends the input analysis object using a polynomial of degree N.

SYNOPSIS ^

function varargout = ltpda_polydetrend(varargin)

DESCRIPTION ^

```
LTPDA_POLYDETREND detrends the input analysis object using a polynomial of degree N.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_POLYDETREND detrends the input analysis object using a
             polynomial of degree N.
CALL:       b = ltpda_polydetrend(a1,a2,a3,..., pl)
INPUTS:     aN  - a list of analysis objects
             pl  - a parameter list
OUTPUTS:    b - array of analysis objects
             If the last input argument is a parameter list (plist) it is used.
             The following parameters are recognised.
PARAMETERS: 'N' - degree of polynomial [default: 1]
VERSION:    $Id: ltpda_polydetrend.m,v 1.11 2008/08/10 20:24:50 hewitson Exp $
HISTORY:    01-03-2007 M Hewitson
             Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_POLYDETREND detrends the input analysis object using a polynomial of degree N.
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_POLYDETREND detrends the input analysis object using a
0006 %              polynomial of degree N.
0007 %
0008 % CALL:       b = ltpda_polydetrend(a1,a2,a3,..., pl)
0009 %
0010 % INPUTS:     aN  - a list of analysis objects
0011 %              pl  - a parameter list
0012 %
0013 % OUTPUTS:    b - array of analysis objects
0014 %
0015 %              If the last input argument is a parameter list (plist) it is used.
0016 %              The following parameters are recognised.
0017 %
0018 % PARAMETERS: 'N' - degree of polynomial [default: 1]
0019 %
0020 %
0021 % VERSION:    $Id: ltpda_polydetrend.m,v 1.11 2008/08/10 20:24:50 hewitson Exp $
```

```
0022 %
0023 % HISTORY:      01-03-2007 M Hewitson
0024 %              Creation
0025 %
0026 %%%%%%%%%%%%%%
0027
0028 function varargout = ltpda_polydetrend(varargin)
0029
0030     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/detrend now.');
```

```
0031
0032     % Call ao/detrend
0033     bs = detrend(varargin{:});
0034
0035     % set output
0036     varargout{1} = bs;
0037 end
0038 % END
```

ltpda_rms

PURPOSE ^

LTPDA_RMS Calculate RMS deviation from spectrum

SYNOPSIS ^

function varargout = ltpda_rms(varargin)

DESCRIPTION ^

```
LTPDA_RMS Calculate RMS deviation from spectrum
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_RMS Calculate RMS deviation from spectrum
CALL:       b=ltpda_rms(a)
PARAMETERS: b - analysis object containing RMS deviation
             a - input analysis object containing spectrum
VERSION:    $Id: ltpda_rms.m,v 1.9 2008/08/10 20:16:49 hewitson Exp $
NOTE:       Taken from code by: 1998.05.25      Masaki Ando
HISTORY:    12-02-07 M Hewitson
             Creation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_RMS Calculate RMS deviation from spectrum
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_RMS Calculate RMS deviation from spectrum
0006 %
0007 % CALL:       b=ltpda_rms(a)
0008 %
0009 % PARAMETERS: b - analysis object containing RMS deviation
0010 %             a - input analysis object containing spectrum
0011 %
0012 % VERSION:    $Id: ltpda_rms.m,v 1.9 2008/08/10 20:16:49 hewitson Exp $
0013 %
0014 % NOTE:       Taken from code by: 1998.05.25      Masaki Ando
0015 %
0016 % HISTORY:    12-02-07 M Hewitson
0017 %             Creation
0018 %
0019 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0020
0021 function varargout = ltpda_rms(varargin)
0022
0023     warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/rms now.');
```

```
0028 % set output
0029 varargout{1} = bs;
0030
0031 end
```

Generated on Mon 08-Sep-2008 13:18:47 by [m2html](#) © 2003

ltpda_xcorr

PURPOSE ^

LTPDA_XCORR makes cross-correlation estimates of the time-series

SYNOPSIS ^

function varargout = ltpda_xcorr(varargin)

DESCRIPTION ^

```
LTPDA_XCORR makes cross-correlation estimates of the time-series
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DESCRIPTION: LTPDA_XCORR makes cross-correlation estimates of the time-series
              objects in the input analysis objects. The cross-correlation is
              computed using MATLAB's xcorr (>> help xcorr).

CALL:        b = ltpda_xcorr(a1,a2,a3,...,pl)

INPUTS:      b      - output analysis objects
              aN     - input analysis objects (at least two)
              pl     - input parameter list

              The function makes correlation estimates between a1 and all other
              input A0s. Therefore, if the input argument list contains N analysis
              objects, the output a will contain N-1 CPSD estimates.

              If only on A0 is input, the auto-correlation is computed.

              If the last input argument is a parameter list (plist) it is used.
              The following parameters are recognised.

PARAMETERS:  'MaxLag'   - compute over range of lags -MaxLag to MaxLag [default: M-1]
              'Scale'   - normalisation of the correlation. Choose from:
                  'biased' - scales the raw cross-correlation by 1/M.
                  'unbiased' - scales the raw correlation by 1/(M-abs(lags)).
                  'coeff'  - normalizes the sequence so that the auto-correlations
                           at zero lag are identically 1.0.
                  'none'   - no scaling (this is the default).

              M is the length of longest input vector. If one vector is shorted,
              it is zero padded.

VERSION:     $Id: ltpda_xcorr.m,v 1.7 2008/08/11 10:01:08 hewitson Exp $

HISTORY:     07-02-2007 M Hewitson
              Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
0001 % LTPDA_XCORR makes cross-correlation estimates of the time-series
0002 %
0003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0004 %
0005 % DESCRIPTION: LTPDA_XCORR makes cross-correlation estimates of the time-series
0006 %              objects in the input analysis objects. The cross-correlation is
0007 %              computed using MATLAB's xcorr (>> help xcorr).
```

```

0008 %
0009 % CALL:      b = ltpda_xcorr(a1,a2,a3,...,pl)
0010 %
0011 % INPUTS:    b      - output analysis objects
0012 %            aN      - input analysis objects (at least two)
0013 %            pl      - input parameter list
0014 %
0015 %            The function makes correlation estimates between a1 and all other
0016 %            input AOs. Therefore, if the input argument list contains N analysis
0017 %            objects, the output a will contain N-1 CPSD estimates.
0018 %
0019 %            If only on AO is input, the auto-correlation is computed.
0020 %
0021 %            If the last input argument is a parameter list (plist) it is used.
0022 %            The following parameters are recognised.
0023 %
0024 % PARAMETERS: 'MaxLag'   - compute over range of lags -MaxLag to MaxLag [default: M-
0025 %               1]
0026 %               'Scale'   - normalisation of the correlation. Choose from:
0027 %               'biased'   - scales the raw cross-correlation by 1/M.
0028 %               'unbiased' - scales the raw correlation by 1/(M-abs(lags)).
0029 %               'coeff'    - normalizes the sequence so that the auto-correlations
0030 %               'none'     - at zero lag are identically 1.0.
0031 %               'none'     - no scaling (this is the default).
0032 %
0033 %            M is the length of longest input vector. If one vector is shorted,
0034 %            it is zero padded.
0035 %
0036 % VERSION:    $Id: ltpda_xcorr.m,v 1.7 2008/08/11 10:01:08 hewitson Exp $
0037 %
0038 % HISTORY:    07-02-2007 M Hewitson
0039 %              Creation
0040 %%%%%%%%%%%
0041
0042 function varargout = ltpda_xcorr(varargin)
0043
0044     warning('!!! This function is deprecated and will be removed from future versions of
0045 LTPDA. Use ao/xcorr now.');
```

```

0046     % Call ao/xcorr
0047     bs = xcorr(varargin{:});
0048
0049     % set output
0050     varargout{1} = bs;
0051 end
0052
0053
```

Constructor examples of the XYDATA class

[Construct empty XYDATA object](#)

[Construct a XYDATA object by loading the object from a file](#)

[Construct a XYDATA object from a y-data vector](#)

[Construct a XYDATA object from a x-data and y-data vector](#)

Construct empty XYDATA object

The following example creates an empty xydata object

```
xy = xydata()
----- xydata 01 -----
name:    None
  x:    [0 0], double
  y:    [0 0], double
xunits:
yunits:
-----
```

Construct a XYDATA object by loading the object from a file

The following example creates a new xydata object by loading the xydata object from disk.

```
xy = xydata('xy.mat')
xy = xydata('xy.xml')
```

Construct a XYDATA object from a y-data vector

Creates an xy data object with the given y-data.

```
y = randn(1000,1);
xy = xydata(y)
```

Construct a XYDATA object from a x-data and y-data vector

creates an xy-data object with the given (x,y)-data.

```
y = randn(1000,1);
x = 1:length(y);
```

```
xy = xydata(x,y)
```

◀ Constructor examples of the TSDATA class Constructor examples of the XYZDATA class ▶

©LTP Team

Constructor examples of the TSDATA class

[Construct empty TSDATA object](#)

[Construct a TSDATA object by loading the object from a file](#)

[Construct a TSDATA object from a data vector](#)

[Construct a TSDATA object from a data vector and time vector](#)

[Construct a TSDATA object from a data vector and the corresponding frequency](#)

Construct empty TSDATA object

The following example creates an empty tsdata object

```
ts = tsdata()
----- tsdata 01 -----
name:  None
fs:    0
x:     [0 0], double
y:     [0 0], double
xunits:
yunits:
nsecs: 0
t0:    1970-01-01 00:00:00.000
-----
```

Construct a TSDATA object by loading the object from a file

The following example creates a new tsdata object by loading the tsdata object from disk.

```
tsd = tsdata('tsd.mat')
tsd = tsdata('tsd.xml')
```

Construct a TSDATA object from a data vector

Creates a time-series object with the given x-data. Sample rate of the data is assumed to be 1Hz.

```
y = randn(1000,1);
tsd = tsdata(y)
```

Construct a TSDATA object from a data vector and time vector

Creates a time-series object with the given (t,x)-data. The sample rate is then set as $1/(t(2)-t(1))$.

```
fs = 10;
x = linspace(0, fs/2, 1000);
y = randn(1000,1);

f1 = tsdata(x,y)
```

Construct a TSDATA object from a data vector and the corresponding frequency

creates a time-series object with the given x-data. The time vector t[] is grown from the sample rate. The first sample is assigned time 0.

```
y = randn(1000,1);
fs = 100;

tsd = tsdata(y,fs)
```

[◀ Constructor examples of the TIMESPAN class](#) [Constructor examples of the XYDATA class ▶](#)

©LTP Team