LTPDA Toolbox

- 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.

Functions - Alphabetical List Getting Started with the LTPDA Toolbox

©LTP Team

Getting Started with the LTPDA Toolbox

What is the LTPDA Toolbox?	Overview of the main functionality of the Toolbox
System Requirements	Supported platforms, MATLAB versions, and required toolboxes.
Setting up MATLAB to work with LTPDA	Installing and editing the LTPDA Startup file.
Starting the LTPDA Toolbox	

©LTP Team

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.

©LTP Team

System Requirements

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

- Platforms
- MATLAB and Related Products
- Additional Programs

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	

Additional Programs

Some of the features in the LTPDA toolbox require additional external programs to be installed.

Graphviz

In order to use the commands listed below, the Graphviz package must be installed.

Method	Description
`history/dotview`	Convert a history object to a tree-diagram using the DOT interpreter.
`ssm/dotview`	Convert the statespace model object to a block-diagram using the DOT interpreter.

The following installation guidelines can be used for different platforms:

Windows

1. Download the relevant package from Downloads section of www.graphviz.org.
2. Install the package by following the relevant instructions.
3. Set the two relevant variables in your `ltpda_startup.m` file:
   - LTPDA_DOT_BIN - set the path to the 'dot.exe' binary. If you perform the default installation, this should be something like:
     LTPDA_DOT_BIN = 'c:\Program Files\Graphviz2.20\bin\dot.exe';
   - LTPDA_DOT_FORMAT - the graphics format to output. See formats for available formats. To view the final graphics file you must have a suitable viewer for that graphics format installed on the system. For example, to output as PDF:
     LTPDA_DOT_FORMAT = 'pdf';

Mac OS X

1. Choose from:
   1. From graphviz:
      1. Download the relevant package from Downloads section of www.graphviz.org.
      2. Install the package by following the relevant instructions.
   2. From Fink:
      1. If you use the fink package manager, in a terminal: `> fink install graphviz`
2. Set the two relevant variables in your `ltpda_startup.m` file:
   - LTPDA_DOT_BIN - set the path to the 'dot' binary. If you perform the default installation from fink, this should be something like:
     LTPDA_DOT_BIN = '/sw/bin/dot';
   - LTPDA_DOT_FORMAT - the graphics format to output. See formats for available formats. To view the final graphics file you must have a suitable viewer for that graphics format installed on the system. For example, to output as PDF:
     LTPDA_DOT_FORMAT = 'pdf';

Linux

1. Choose from:
   1. From graphviz:
      - Download the relevant package from Downloads section of www.graphviz.org.
      - Install the package by following the relevant instructions.
   2. From terminal (Ubuntu):
      - Please type in a terminal: `>sudo apt-get install graphviz`
   3. From graphical package manager like **YaSt, Synaptic, Adept, ...**

        - Start your graphical package manager
        - Search for the `>graphviz` package
        - Select the package and all depending packes and install these packages.
2. Set the two relevant variables in your `ltpda_startup.m` file:
    - LTPDA_DOT_BIN - set the path to the 'dot' binary. If you perform the default installation from the terminal, this should be something like:
      LTPDA_DOT_BIN = '/usr/bin/dot';
      LTPDA_DOT_BIN - even 'dot' without the path should work
      LTPDA_DOT_BIN = 'dot';
    - LTPDA_DOT_FORMAT - the graphics format to output. See formats for available formats. To view the final graphics file you must have a suitable viewer for that graphics format installed on the system. For example, to output as PDF:
      LTPDA_DOT_FORMAT = 'pdf';
3. Define a programm in MATLAB which opens the file.
    - The default programm to open a pdf file is the Acrobat Reader
    - Define another program under File -> Preferences -> Help -> PDF Reader

What is the LTPDA Toolbox | Setting-up MATLAB

©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: `>> java.util.TimeZone.getAvailableIDs`
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 functionalty 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`.

Setting-up MATLAB Trouble-shooting

©LTP Team

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.

©LTP Team

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

©LTP Team

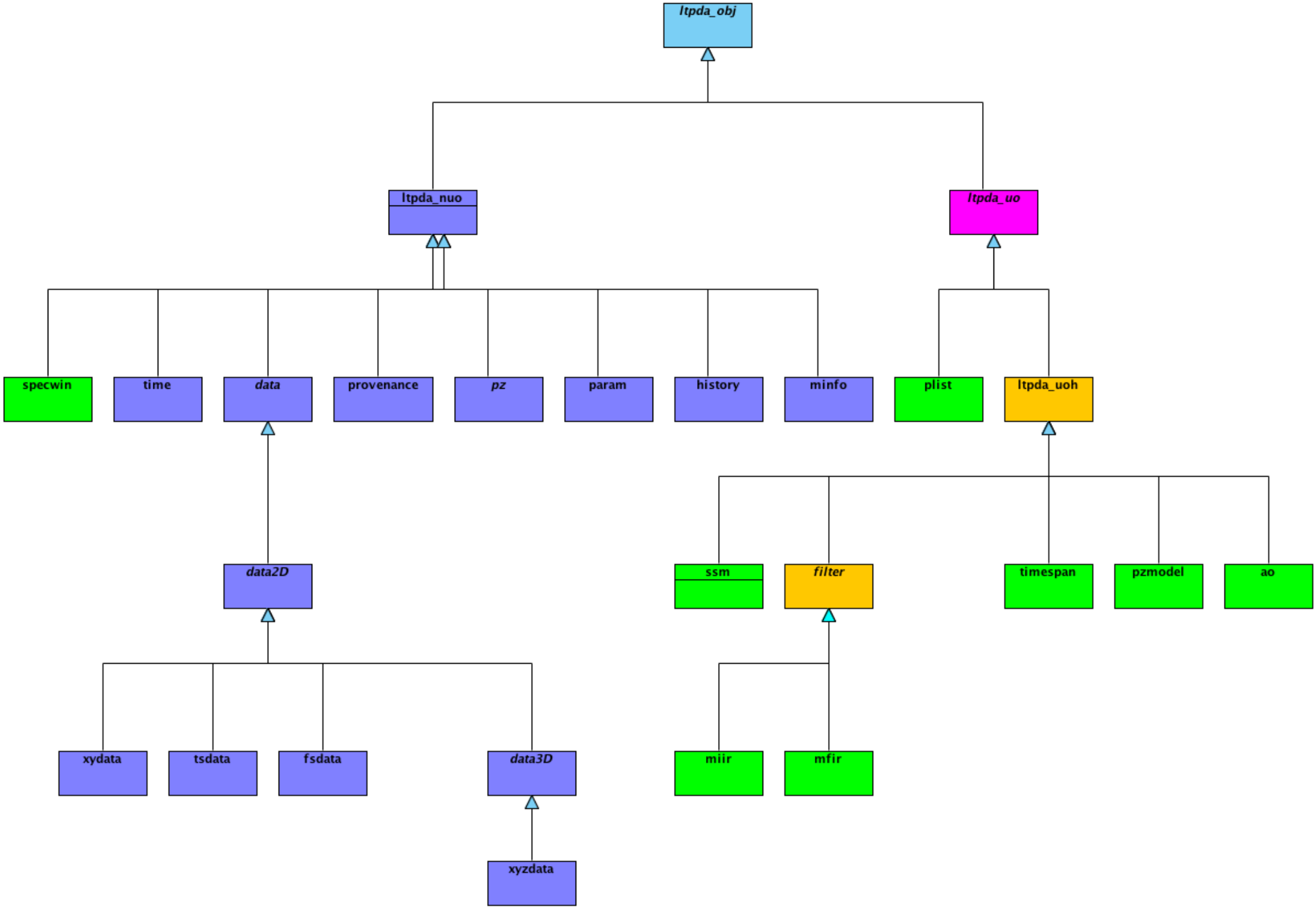
Introducing LTPDA Objects

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`.


ltpda_obj
ltpda_nuo
ltpda_uo
specwin
time
data
provenance
pz
param
history
minfo
plist
ltpda_uoh
data2D
ssm
filter
timespan
pzmodel
ao
xydata
tsdata
fsdata
data3D
miir
mfir
xyzdata

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

©LTP Team

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.

Introducing LTPDA Objects Working with LTPDA objects

©LTP Team

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`.

Creating LTPDA Objects Analysis Objects

©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.

Working with LTPDA objects Creating Analysis Objects

©LTP Team

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)
 provenance:  created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
(R2007a) 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)
 provenance:  created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
(R2007a) 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
 provenance:  created by unknown@bob.local[192.168.2.100] on MACI/7.4 (R2007a)/0.2a
(R2007a) 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.

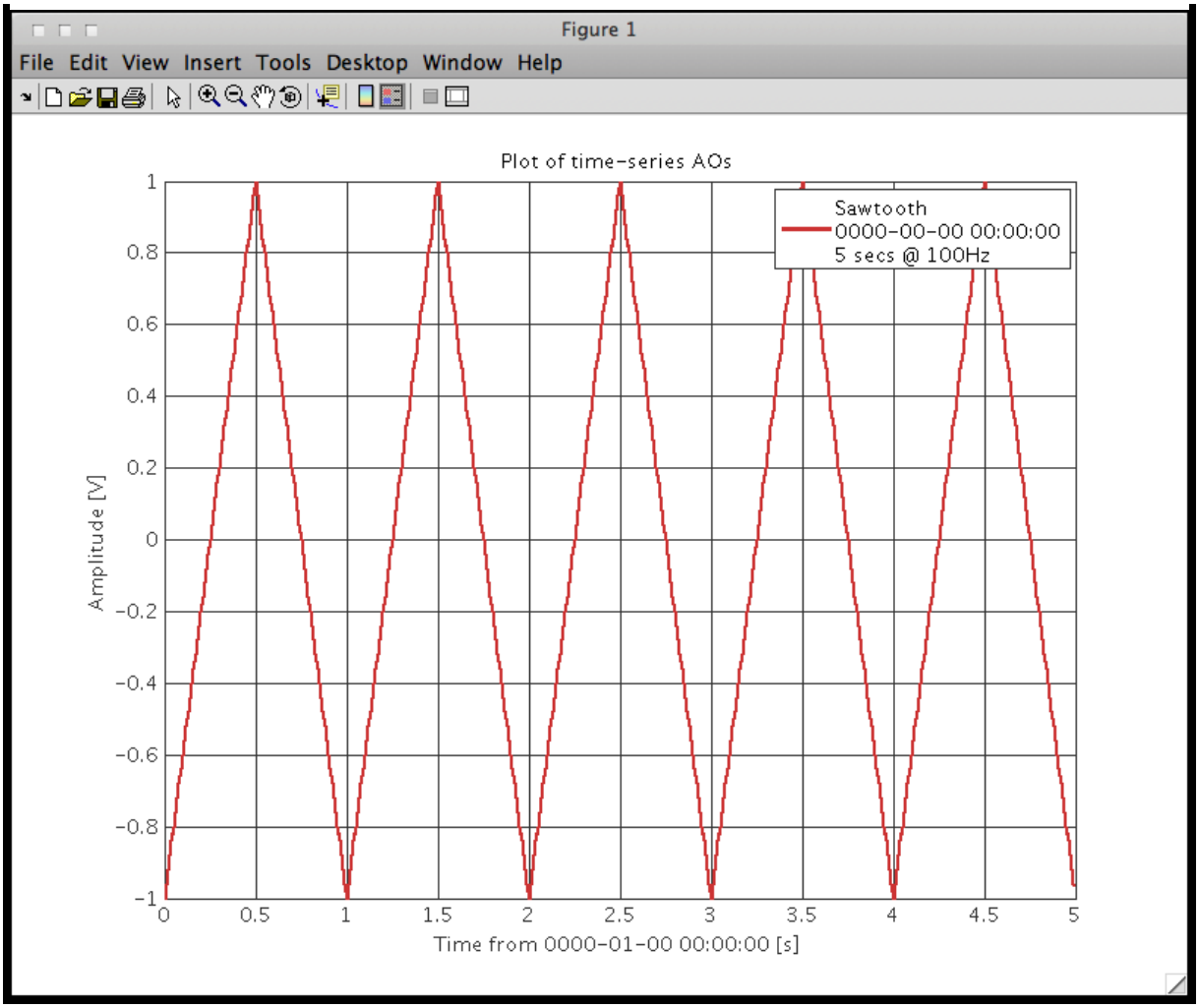
Creating AOs from waveform descriptions

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 `iplot` function to view the resulting waveform:

```
iplot(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
```

Converting existing data into Analysis Objects Plotting Analysis Objects

©LTP Team

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:

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

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.

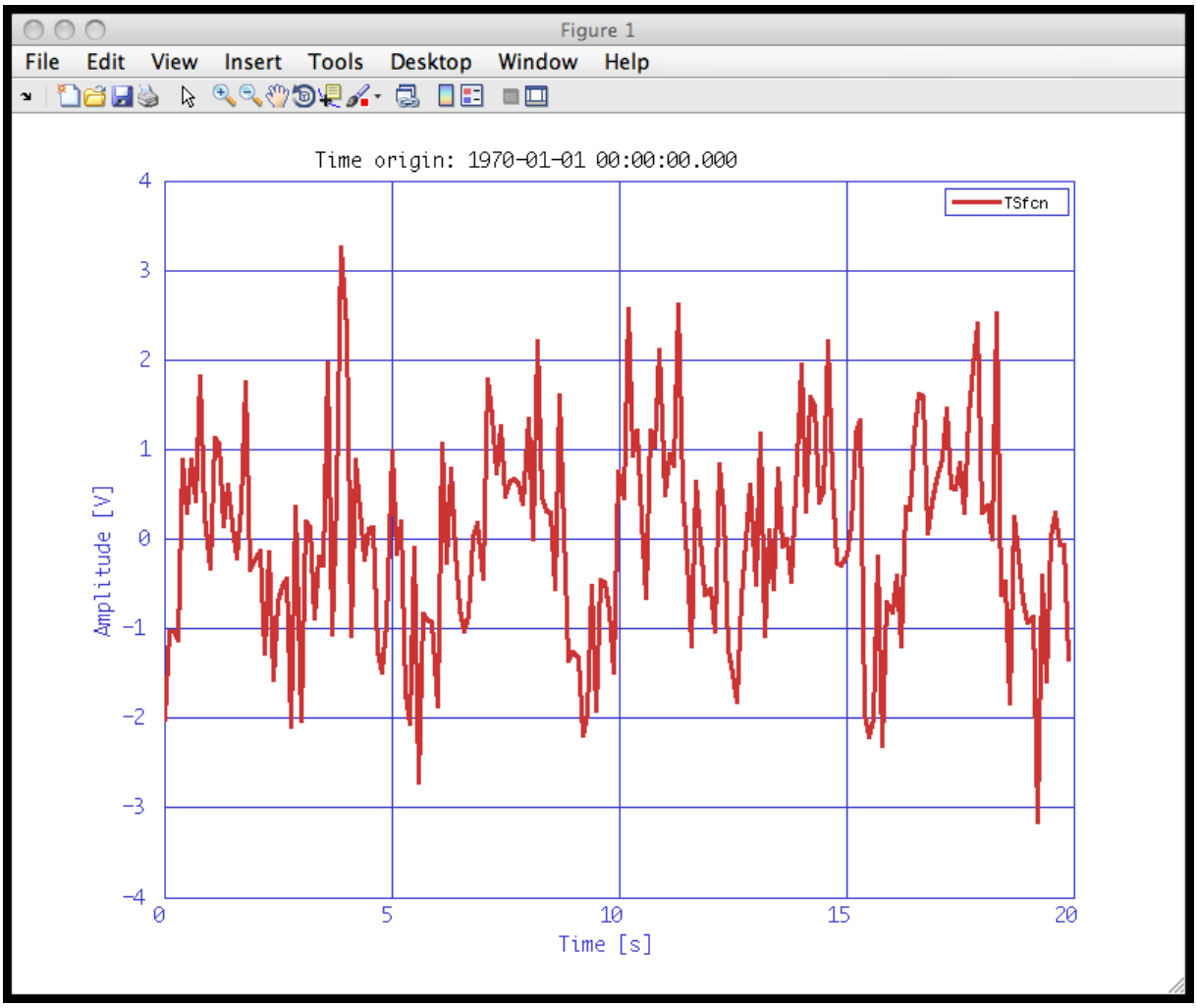
In addition, `iplot` can be configured using a input `plist`. The following examples show some of the possible ways to use `iplot`

```
>> 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 `iplot` we see these units reflected in the x and y axis labels.

```
>> iplot(a1)
```



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

title.

Saving Analysis Objects Parameter Lists

©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

Plotting Analysis Objects Creating Parameters

©LTP Team

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 parmeter object can be accessed as follows:

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


©LTP Team

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

Creating lists of Parameters What are LTPDA spectral windows?

©LTP Team

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 PSLL.

Spectral Windows Create spectral windows

©LTP Team

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:

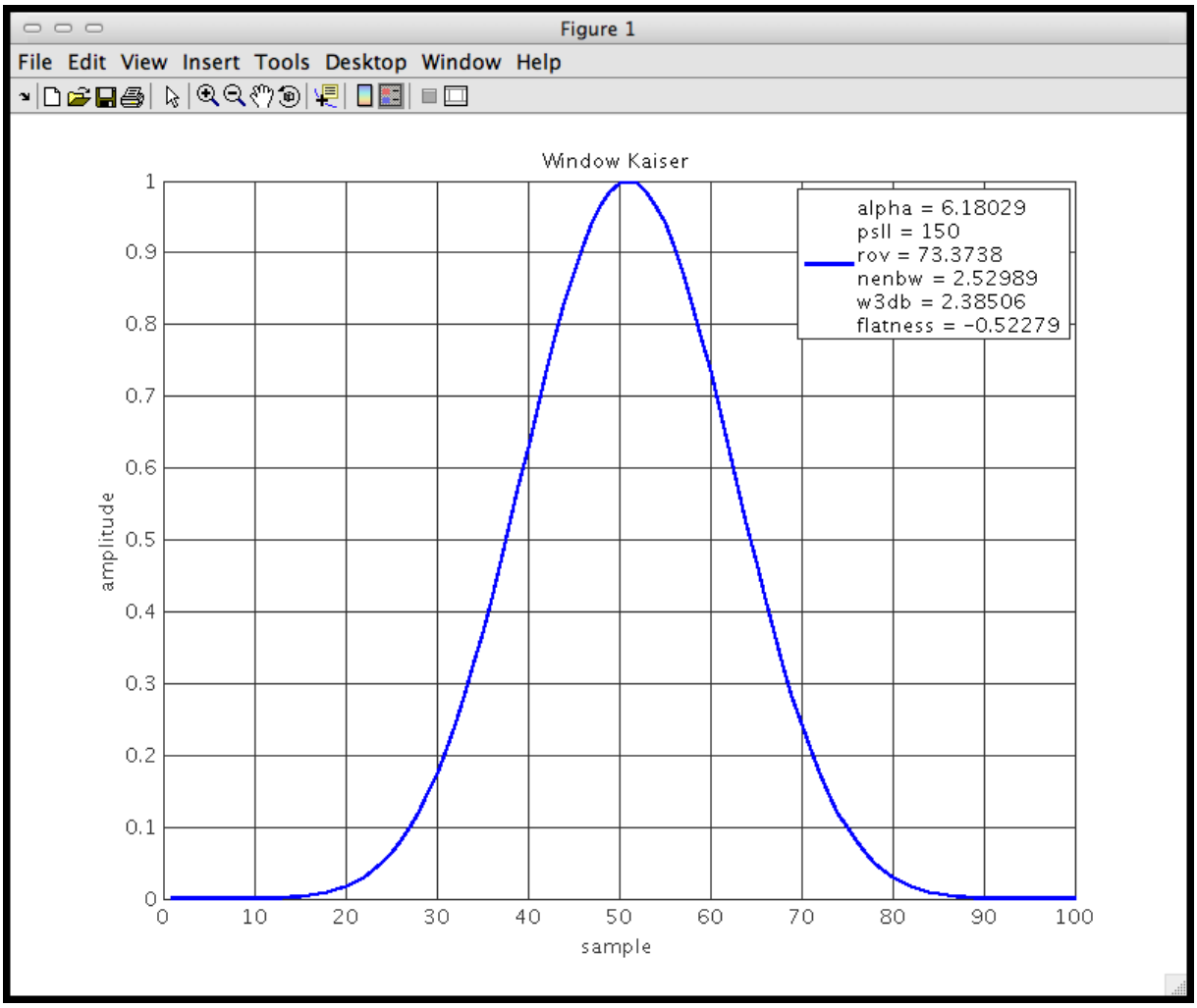
```
>> w = specwin('Kaiser', 100, 150)
-------- Kaiser ------------

   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

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


©LTP Team

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.

©LTP Team

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`.

Visualising spectral windows Simulation/modelling

©LTP Team

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 is 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.

©LTP Team

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 `T = 1/fs`, where `fs` is the sampling frequency.

- y0 = Tinit * r, on initialization
- yi = E * yi-1 + Tprop * r, to propagate
- xi = a * yi , the sampled time series.

r is a vector of independent normal Gaussian random numbers Tinit, E, Tprop 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)

- fs - 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:
  - Tinit - matrix to calculate initial state vector
  - Tprop - 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 `fs`. 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
  - Tinit
  - Tprop
  - E
- the coefficients of transfer function:
  - num
  - den
- the input parameters
  - gain
  - fs
  - nsecs

Case 2: Starting from given matrices

The matrices `Tinit, Tprop 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_noisegen.
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_noisegen 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

Franklin noise-generator Creating poles and zeros

©LTP Team

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.

©LTP Team

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:

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

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

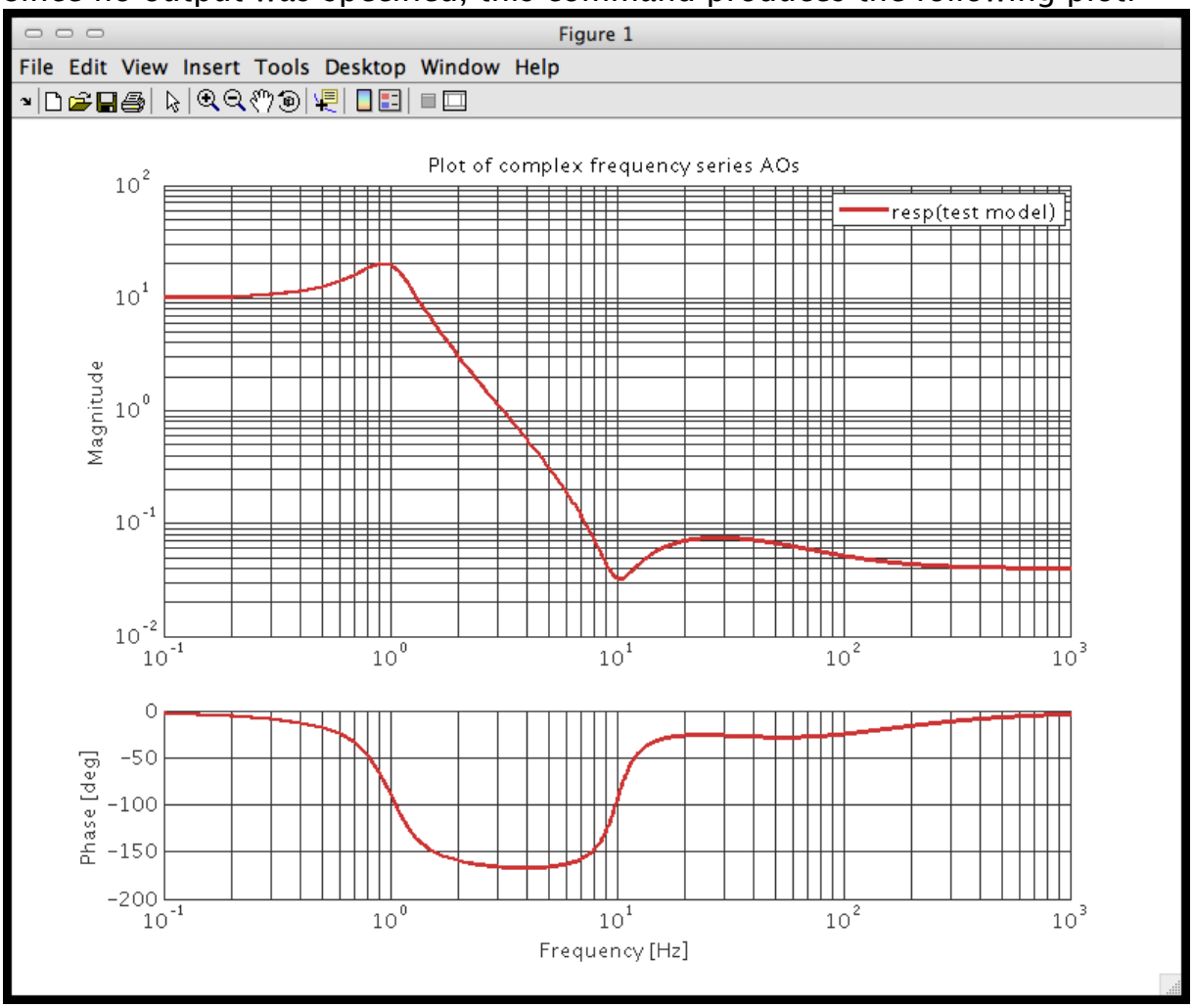
```
>> pl  = plist('name', 'test model', ...
               'gain', 10, ...
            'poles', [pole(1,2) pole(40)], ...
            'zeros', [zero(10,3) zero(100)]);
>> pzm = pzmodel(pl)
---- pzmodel 1 ----
model: test model
gain: 10
pole 001: pole(1,2)
pole 002: pole(40)
zero 001: zero(10,3)
zero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

Computing the response of the model

The frequency response of the model can 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 `iplot` function.

Creating poles and zeros Model helper GUI

©LTP Team

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.

Building a model Converting models to IIR filters

©LTP Team

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.

©LTP Team

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

Converting models to IIR filters Downsampling data

©LTP Team

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)
```

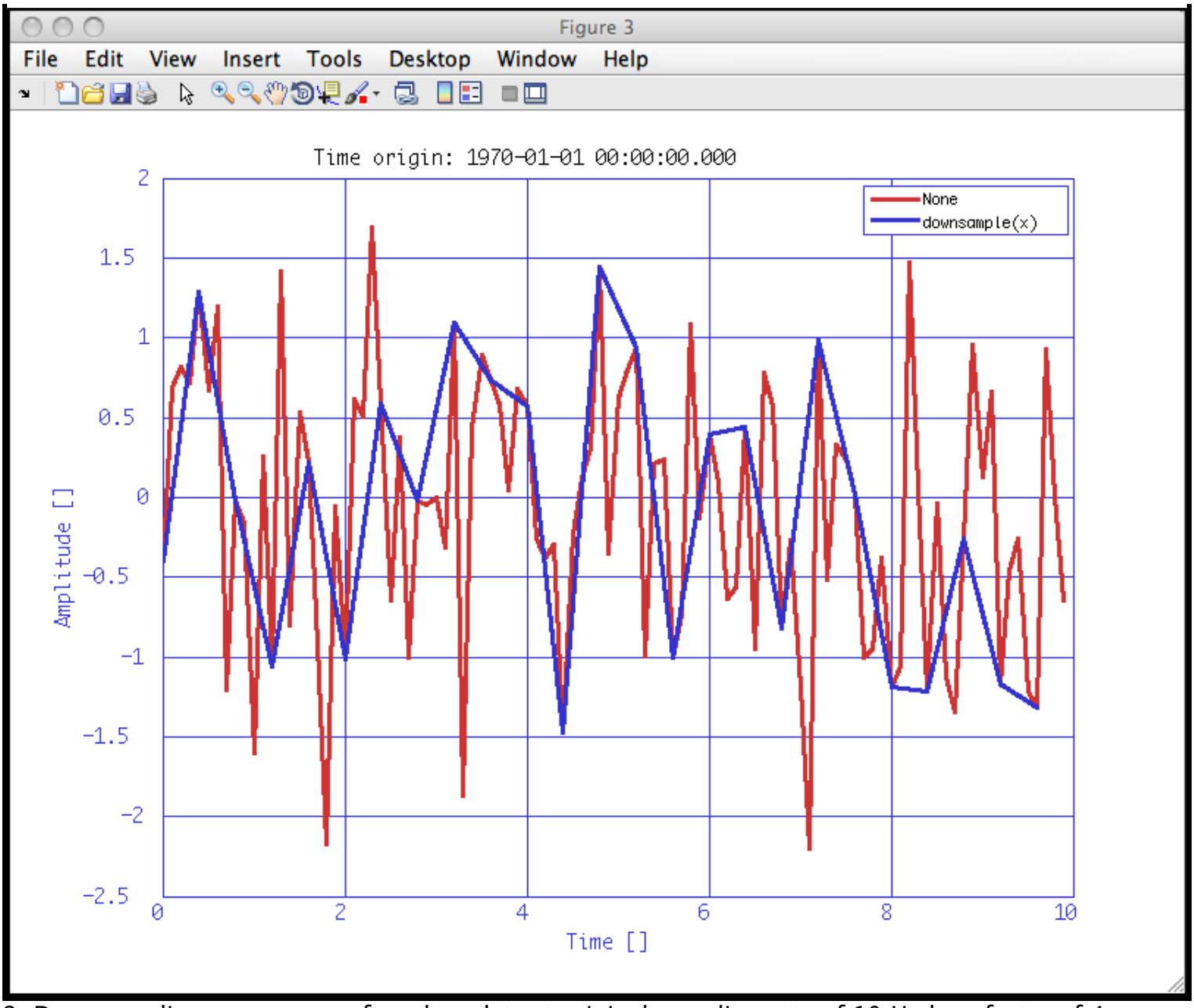
With the following parameters:

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

Examples

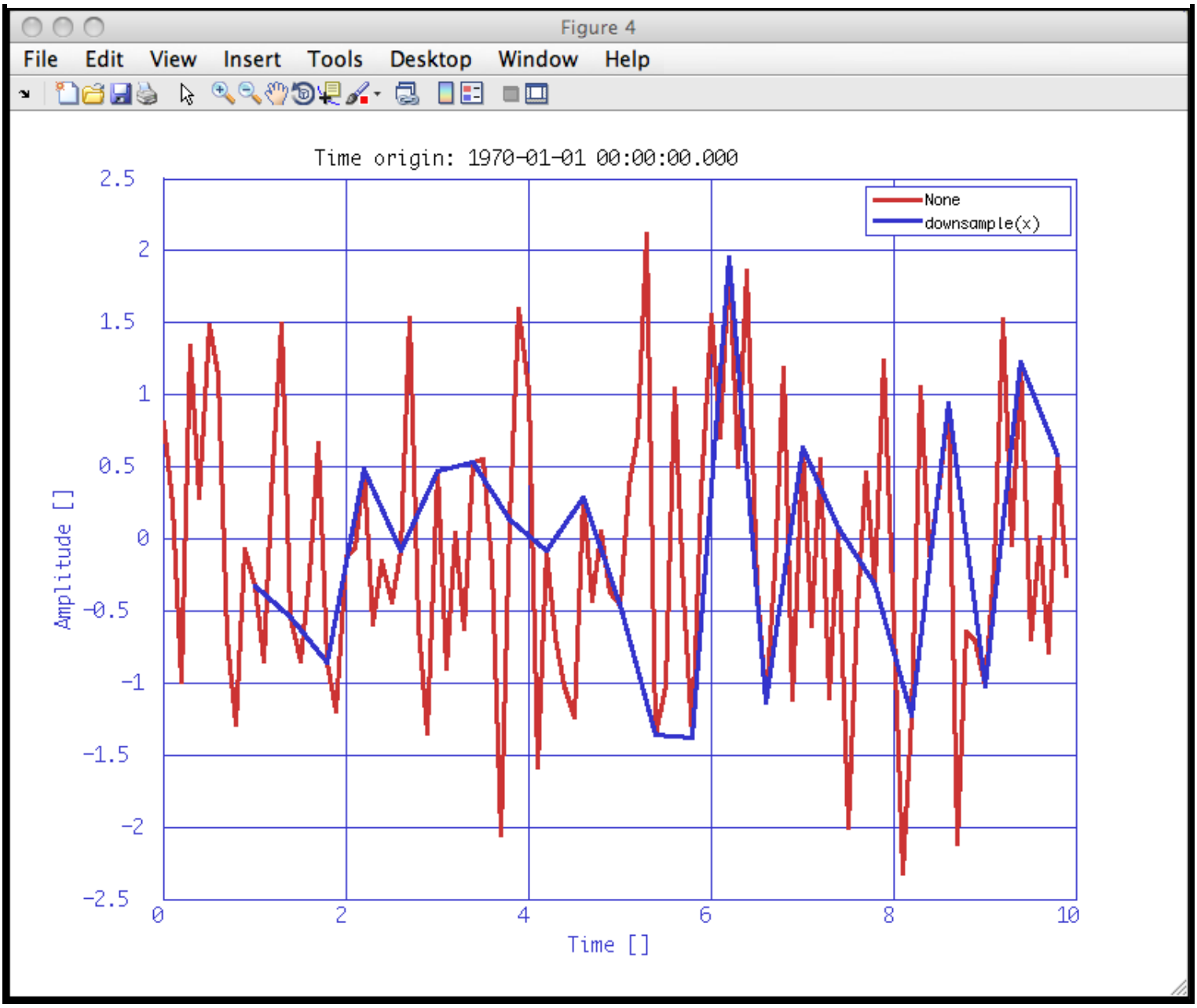
1. Downsampling a sequence of random data at original sampling rate of 10 Hz by a factor of 4 (`fsout` = 2.5 Hz) and no `offset`.

```
x=ao(tsdata(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 (`fsout` = 2.5 Hz) and `offset` = 10.

```
x=ao(tsdata(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
pl = append(pl, param('offset', 10)); % add the offset parameter
y = downsample(x, pl); % downsample the input AO, x
iplot(x, y) % plot original,x, and decimated,y, AOs
```



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*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)
```

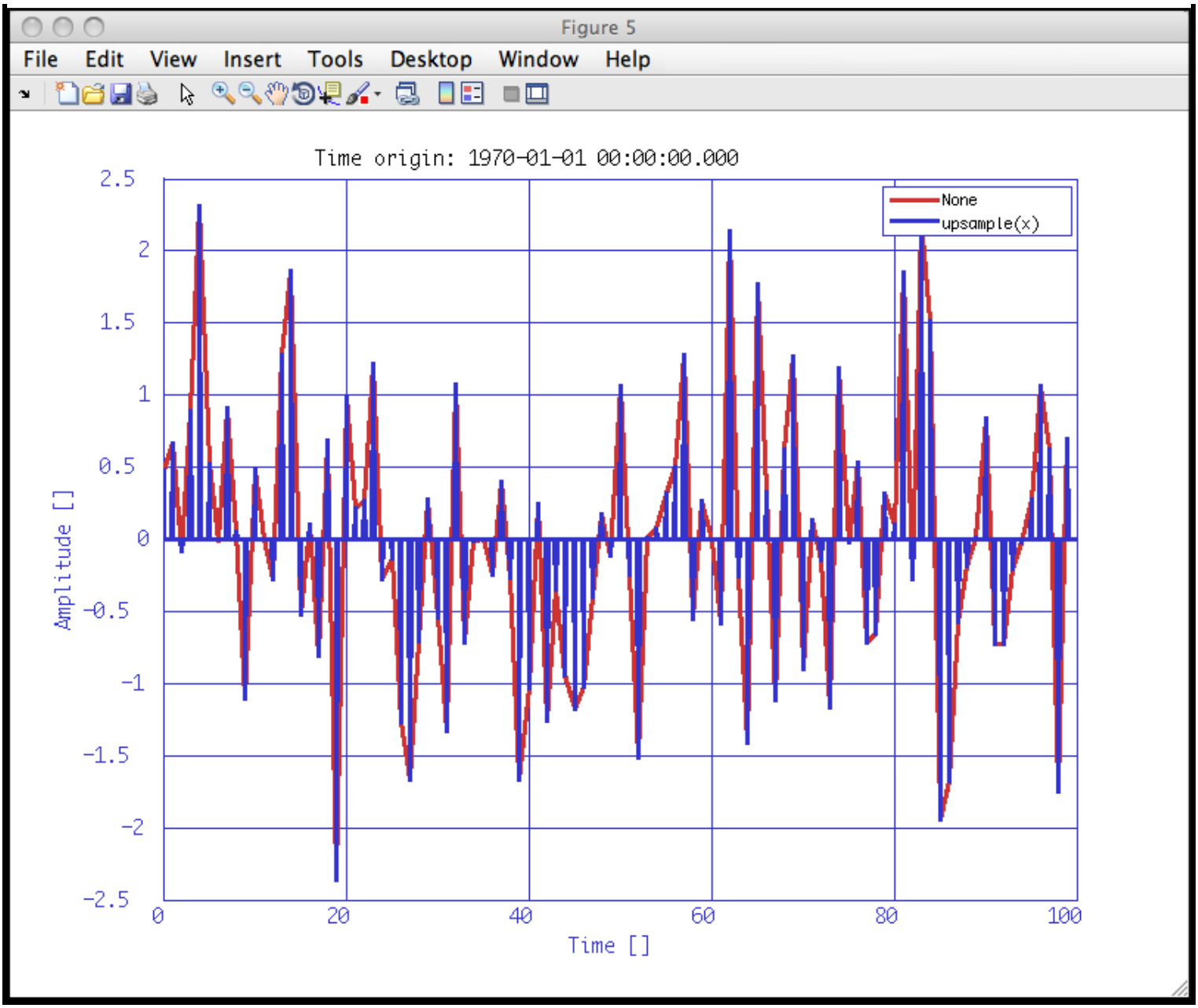
With the following parameters:

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

Examples

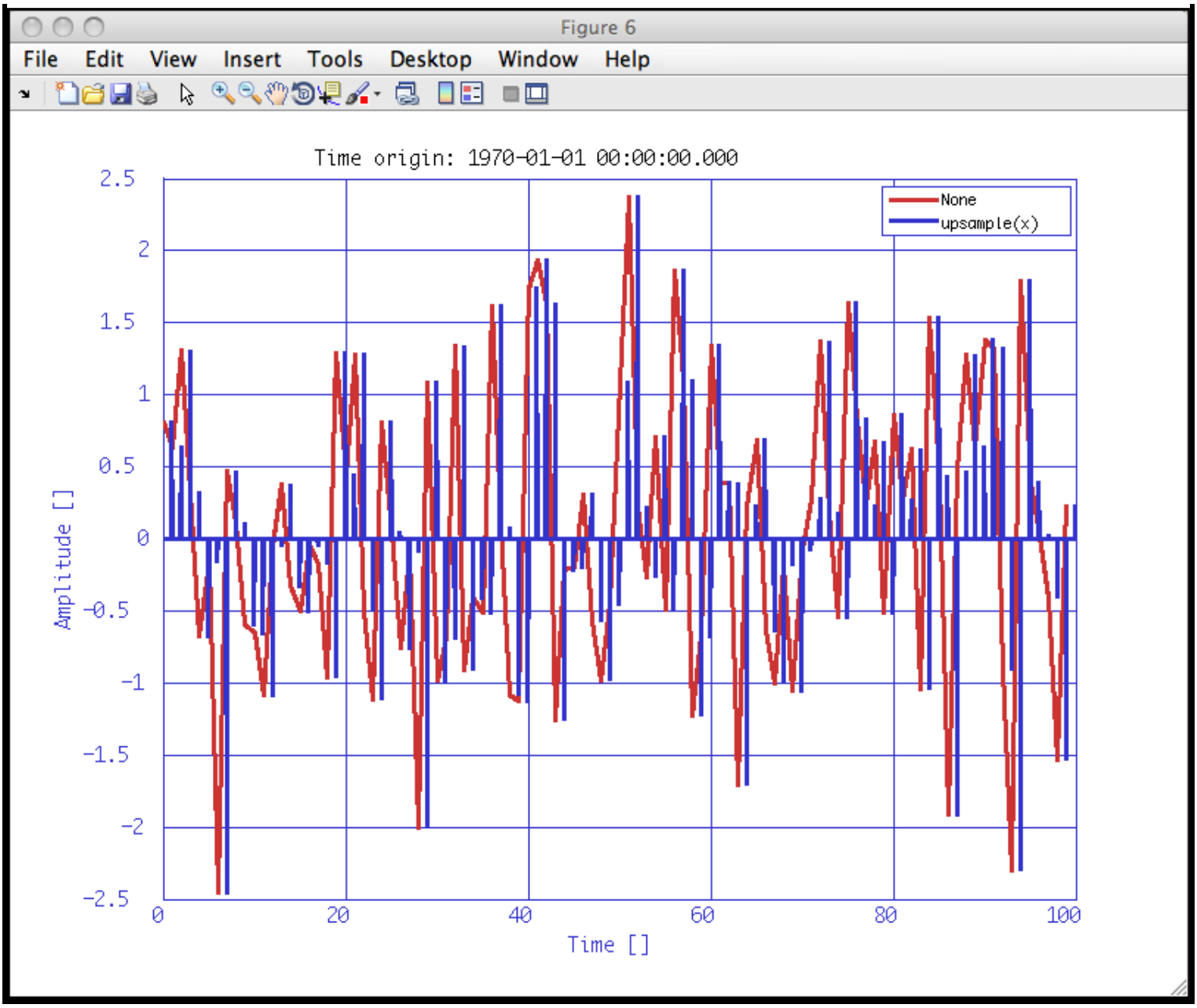
1. Upsampling a sequence of random data at original sampling rate of 1 Hz by a factor of 10 with no initial phase.

```
x=ao(tsdata(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
10
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.

```
x=ao(tsdata(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 21

pl = append(pl, param('phase', 20)); % add phase of 20 samples to the upsampled data

y = upsample(x, pl); % resample the input AO (x) to obtain the upsampled and delayed AO (y)

iplot(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)
```

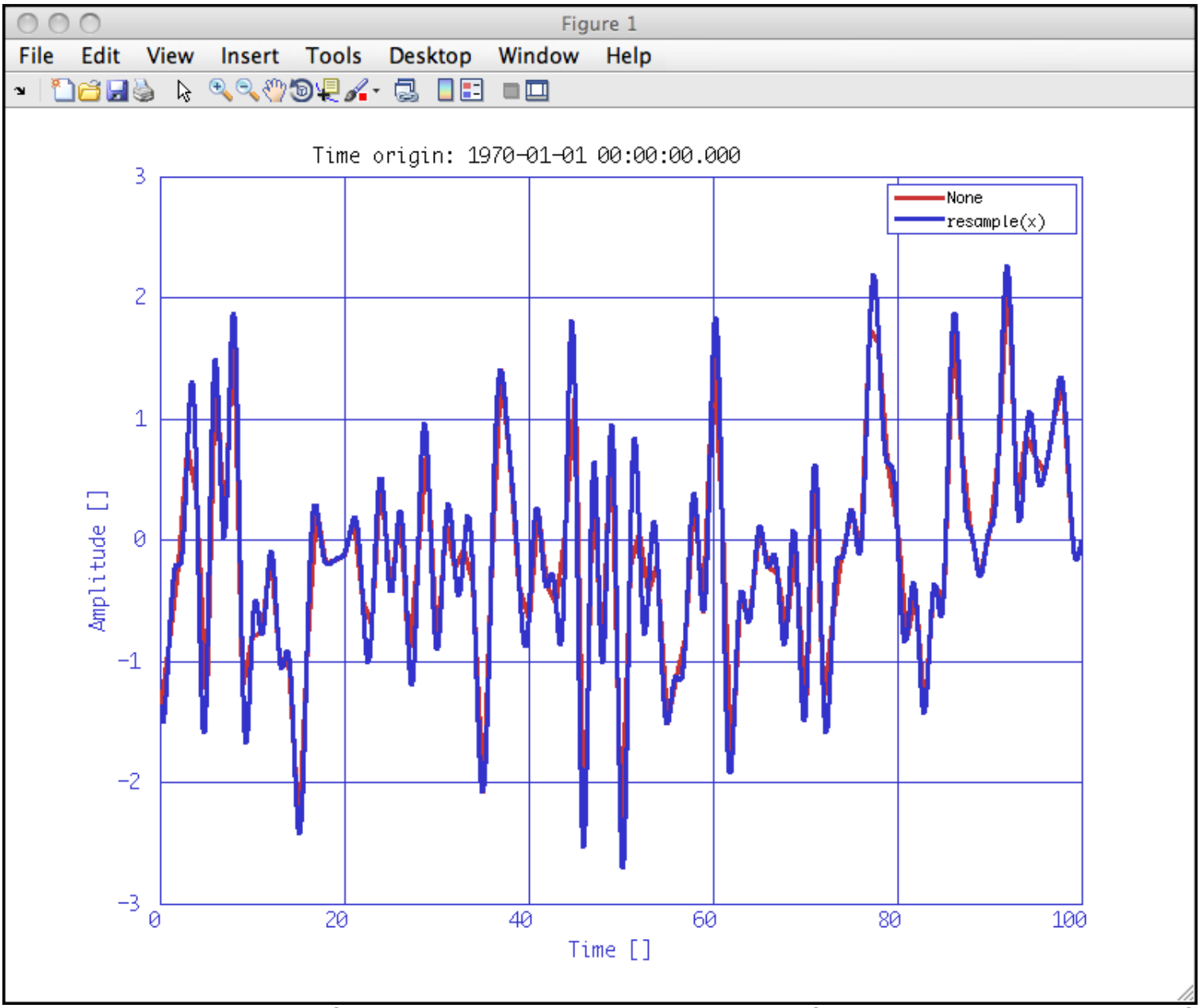
With the following parameters:

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

Examples

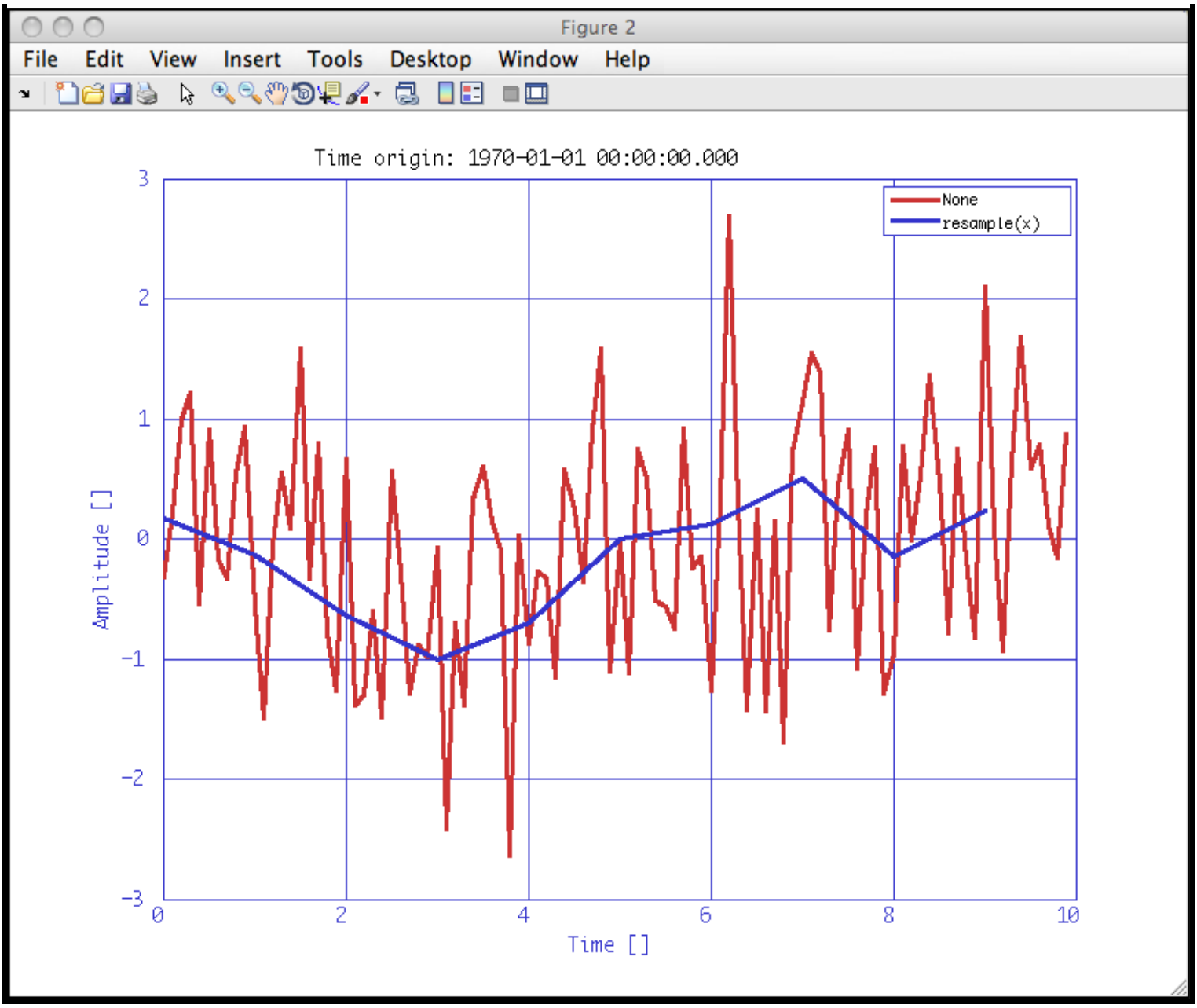
1. Resampling a sequence of random data at original sampling rate of 1 Hz at an output sampling of 50 Hz.

```
                x=ao(tsdata(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
(y)
                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(tsdata(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
pl = append(pl, param('filter', filter)); % use the defined filter in the
resampling process
y = resample(x, pl);  % resample the input AO (x) to obtain the resampled output
AO (y)
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:

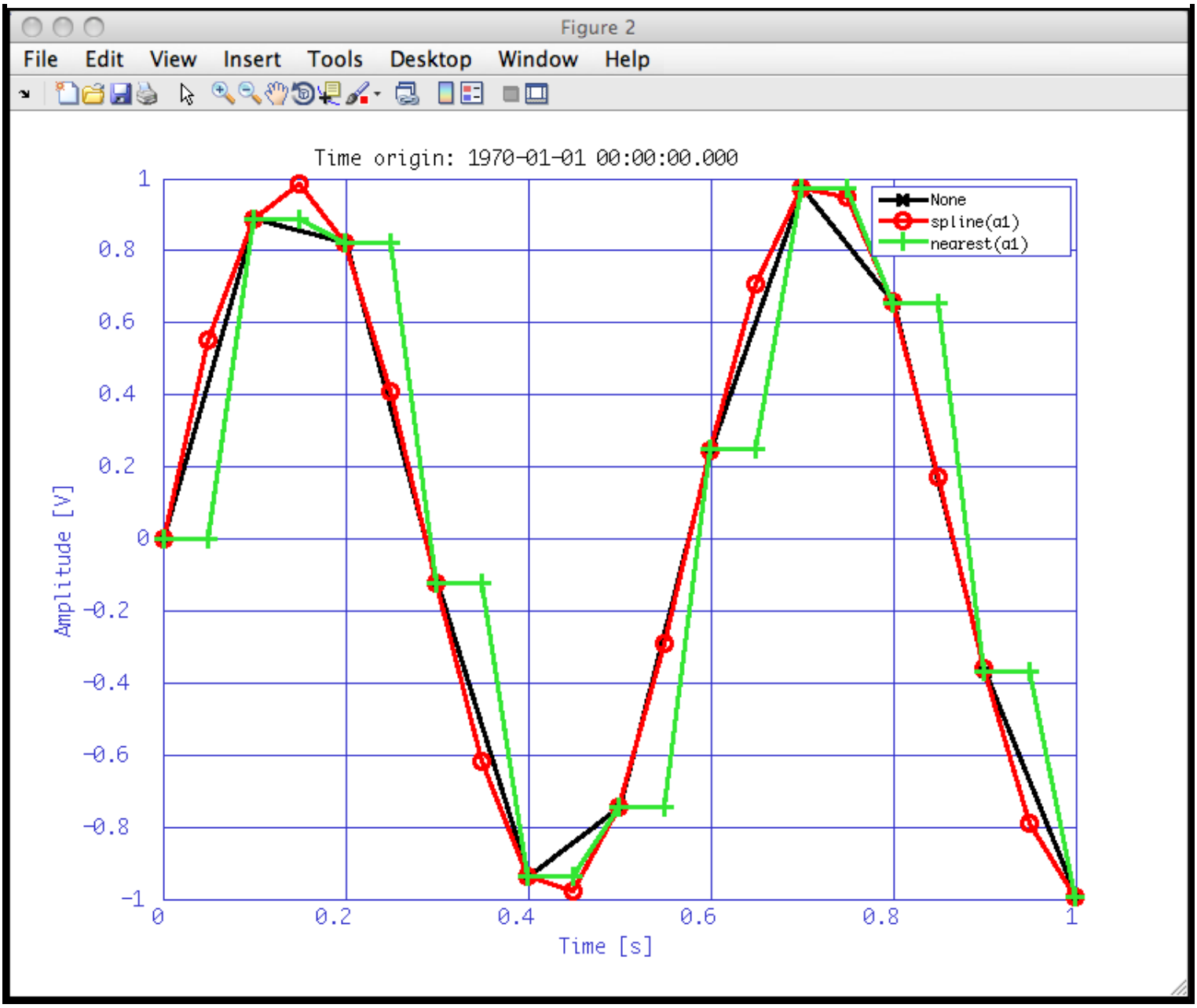
- `'vertices'` – specify the new vertices to interpolate on
- `'method'` – four methods are available for interpolating data
  - 'nearest'– nearest neighbor interpolation
  - 'linear' – linear interpolation
  - 'spline' – spline interpolation (default option)
  - 'cubic' – shape-preserving piecewise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

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

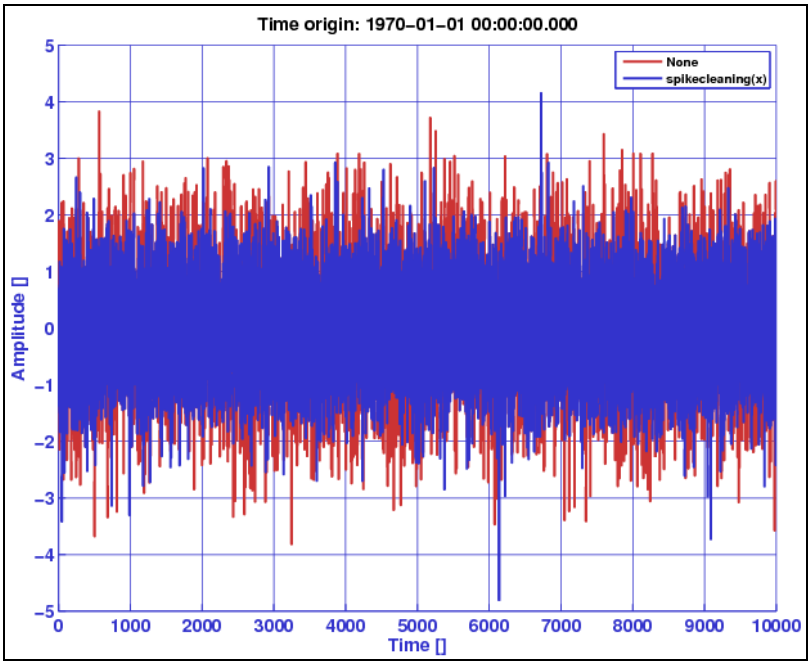
3. `Previous` : the spike is substitued by the previous sample, i.e.:

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

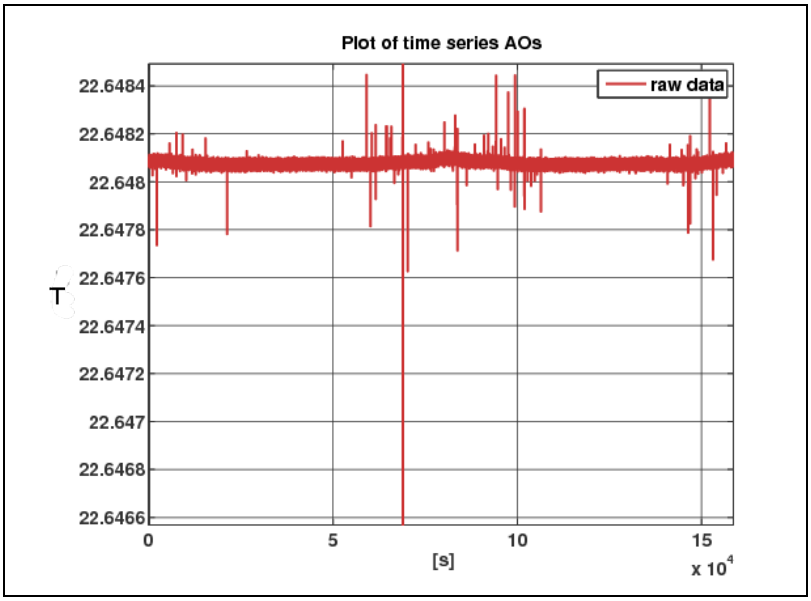
Examples

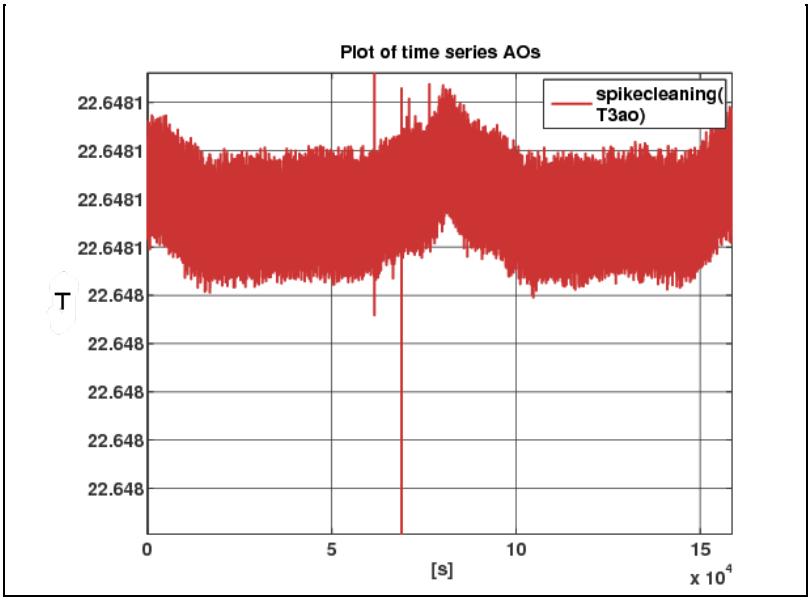
1. Spike cleaning of a sequence of random data with `kspike=2`.

```
Hz.
                    x = ao(tsdata(randn(10000,1),1)); % create an AO of random data sampled at 1

                    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 supression.





Interpolating data

Data gap filling

©LTP Team

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)
```

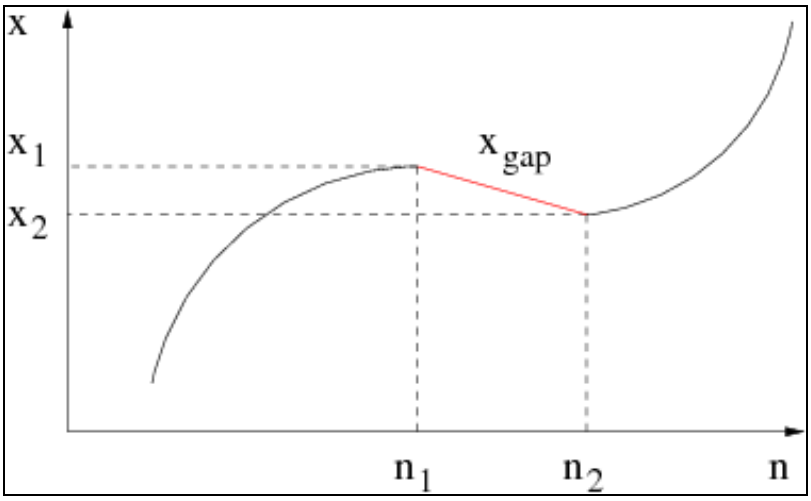
where `a1` and `a2` are the two segments to joint. The parameters are:

- `'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` :

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

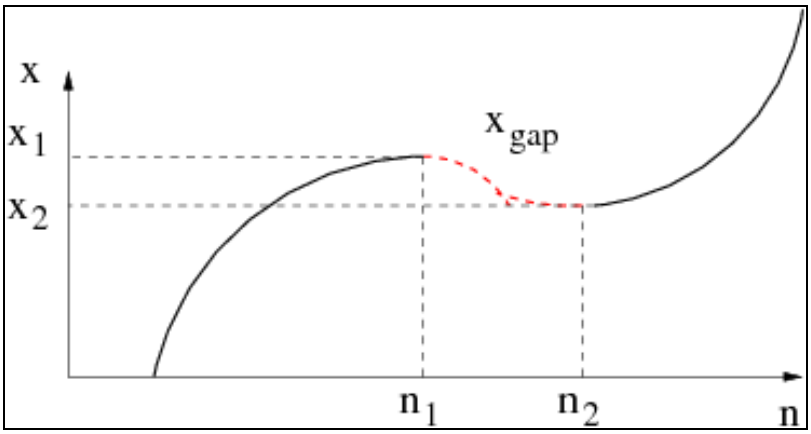


2. `Spline (or third order interpolation)` :

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

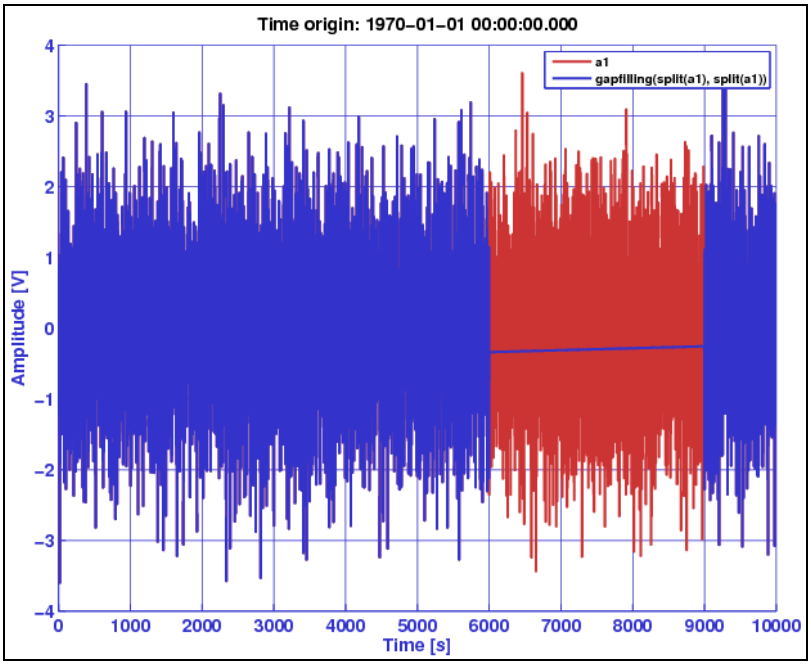
The parameters `a`, `b`, `c` and `d` are calculated by solving next system of equations:

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

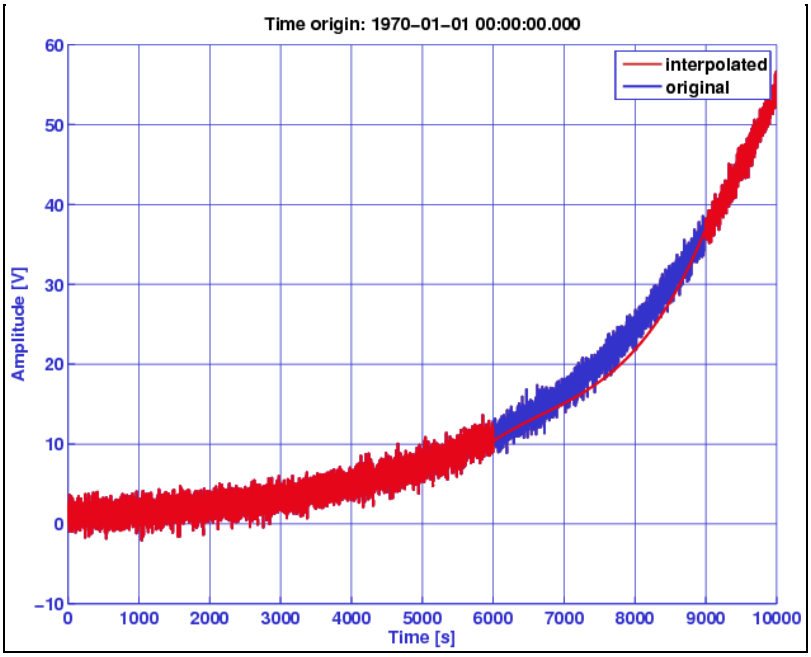


Examples

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.



Spikes reduction in data

Signal Processing in LTPDA

©LTP Team

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

Data gap filling Digital Filtering

©LTP Team

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:

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

  AR processes can be also classified as IIR Filters.

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

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

  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:

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

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

Notice that the convetion 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:

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

```
>> 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 Heinzel, Measurement 39 (2006) 120-129).

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

- `LTPDA_LPSD` Log-scale power spectral density estimates
- `LTPDA_LCPSD` Log-scale cross-spectral density estimates
- `LTPDA_LCOHERENCE` Log-scale magnitude squared coherence estimates
- `LTPDA_LTFE` Log-scale transfer function estimates

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

©LTP Team

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 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. 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

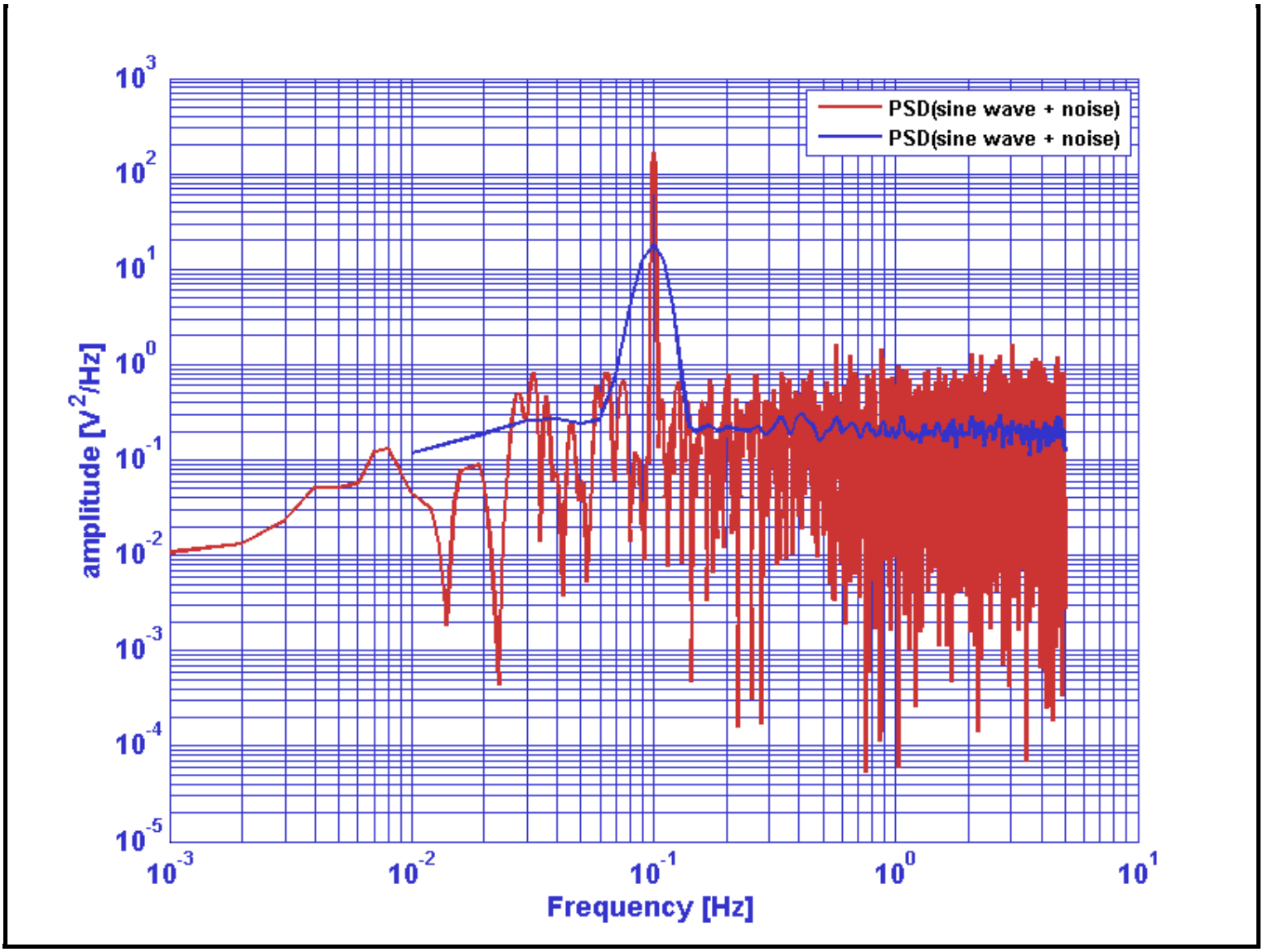
The length of the window is set by the value of the parameter `'Nfft'`, so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

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

Examples

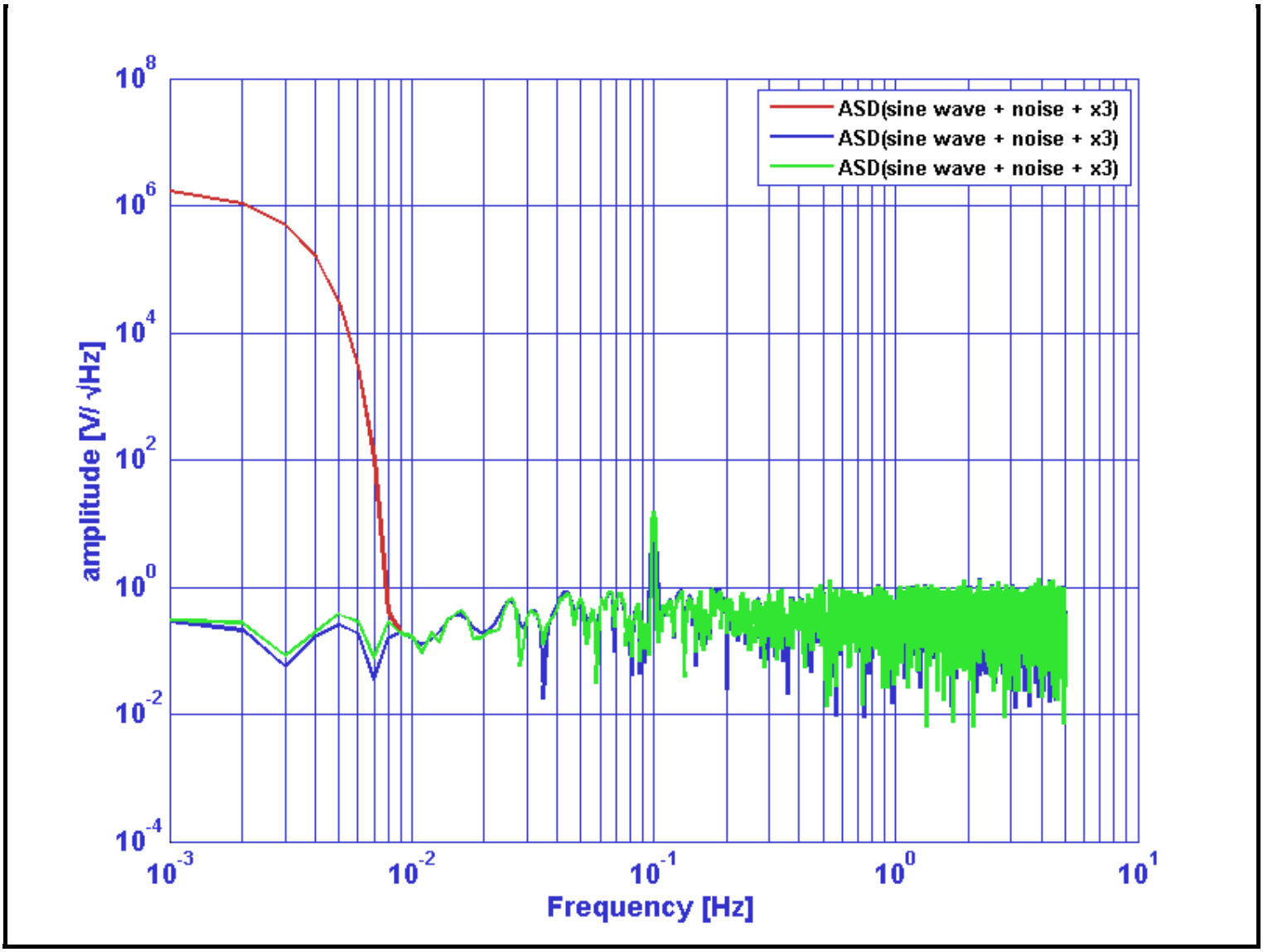
1. Evaluation of the PSD of a time-series represented by a low frequency sinewave signal, superimposed to white noise. Comparison of the effect of windowing on the estimate of the white noise level and on resolving the signal.

```
    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));
    iplot(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 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_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

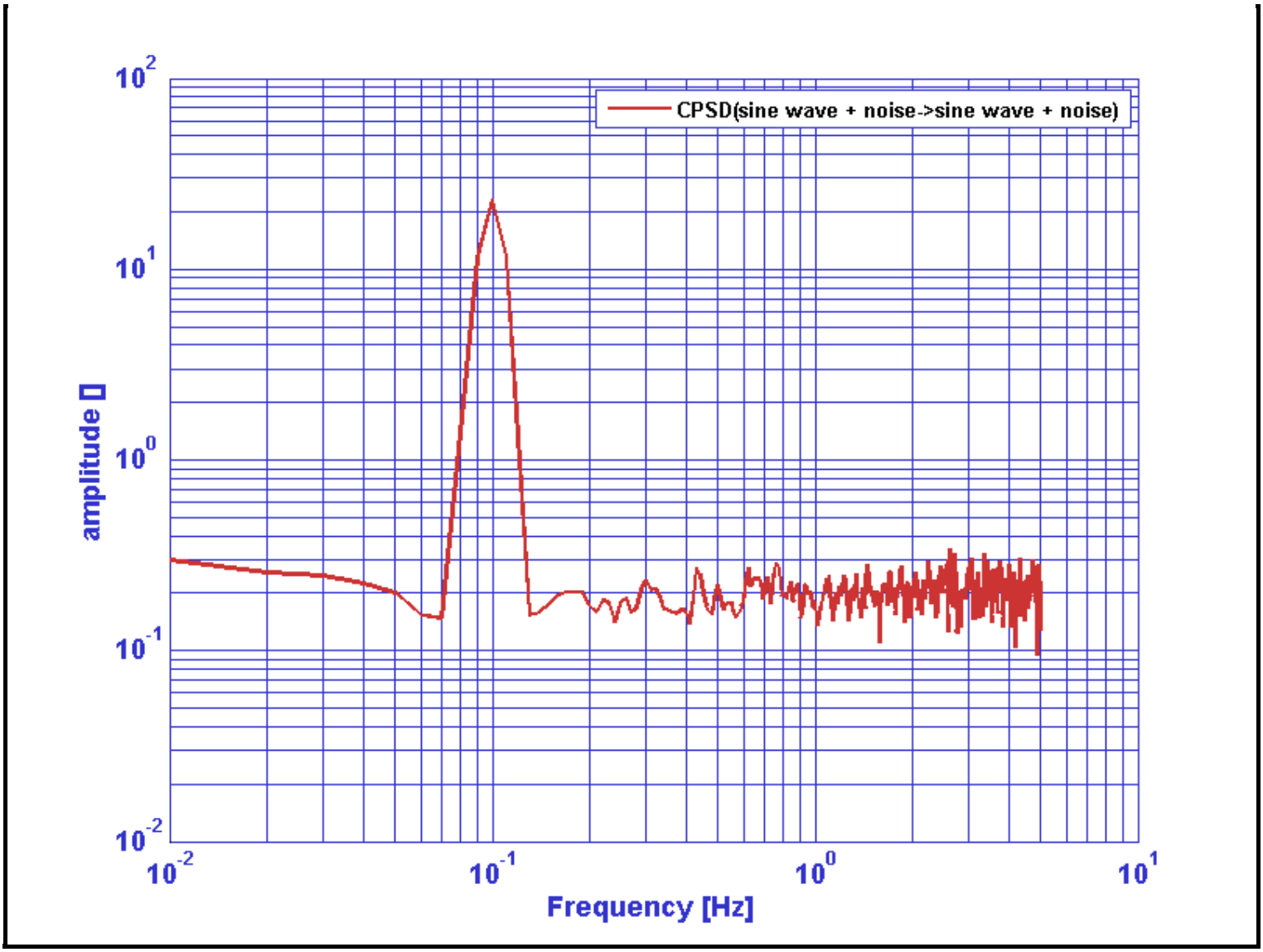
The length of the window is set by the value of the parameter `'Nfft'`, so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

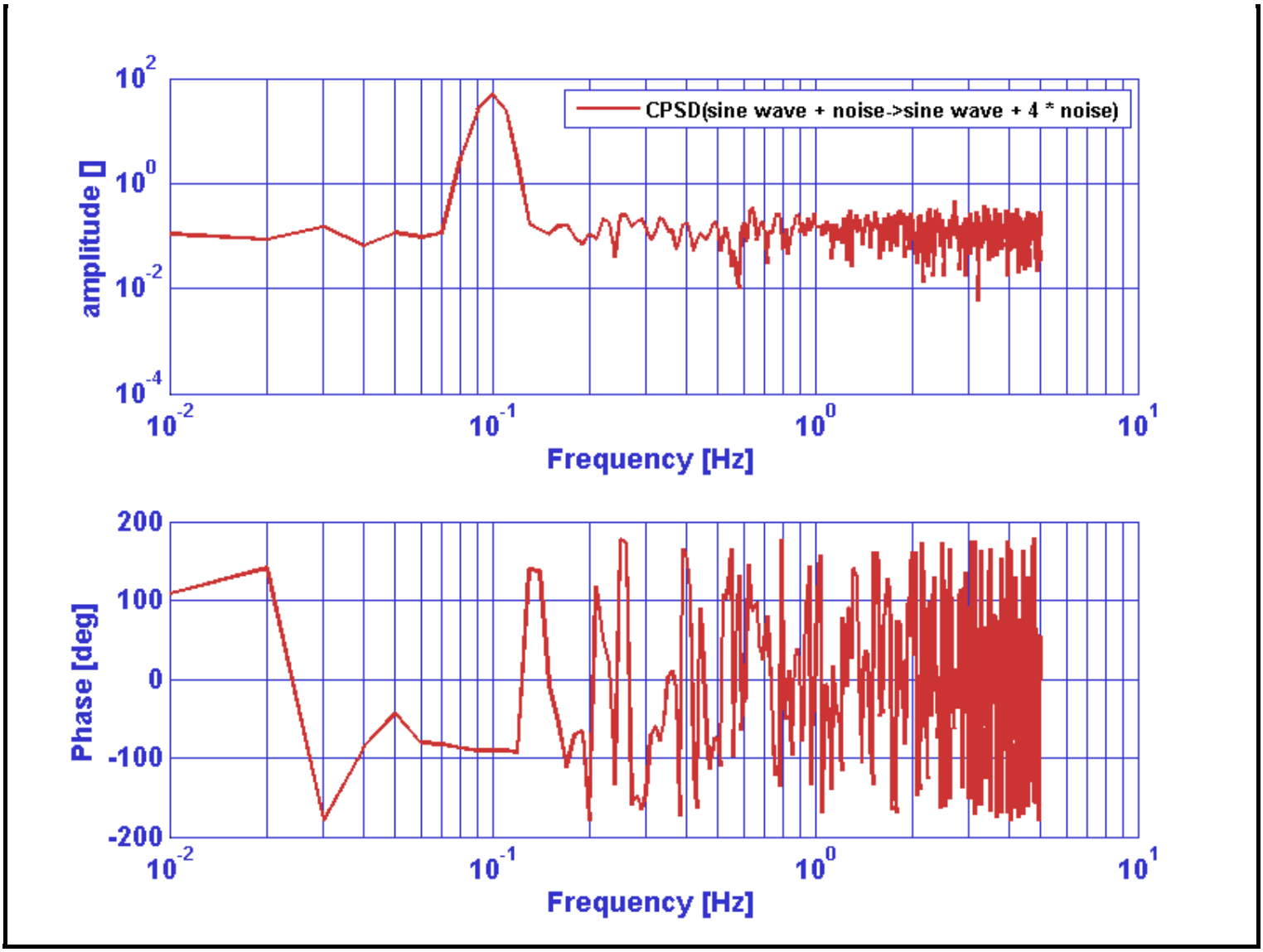
```
  If the user doesn't specify the value of a given parameter, the default value is used.
  The function makes CPSD estimates between all input AOs. Therefore, if the input argument
list contains N analysis objects, the output a will contain NxN CPSD estimates.
The diagonal elements will be S_ai_ai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

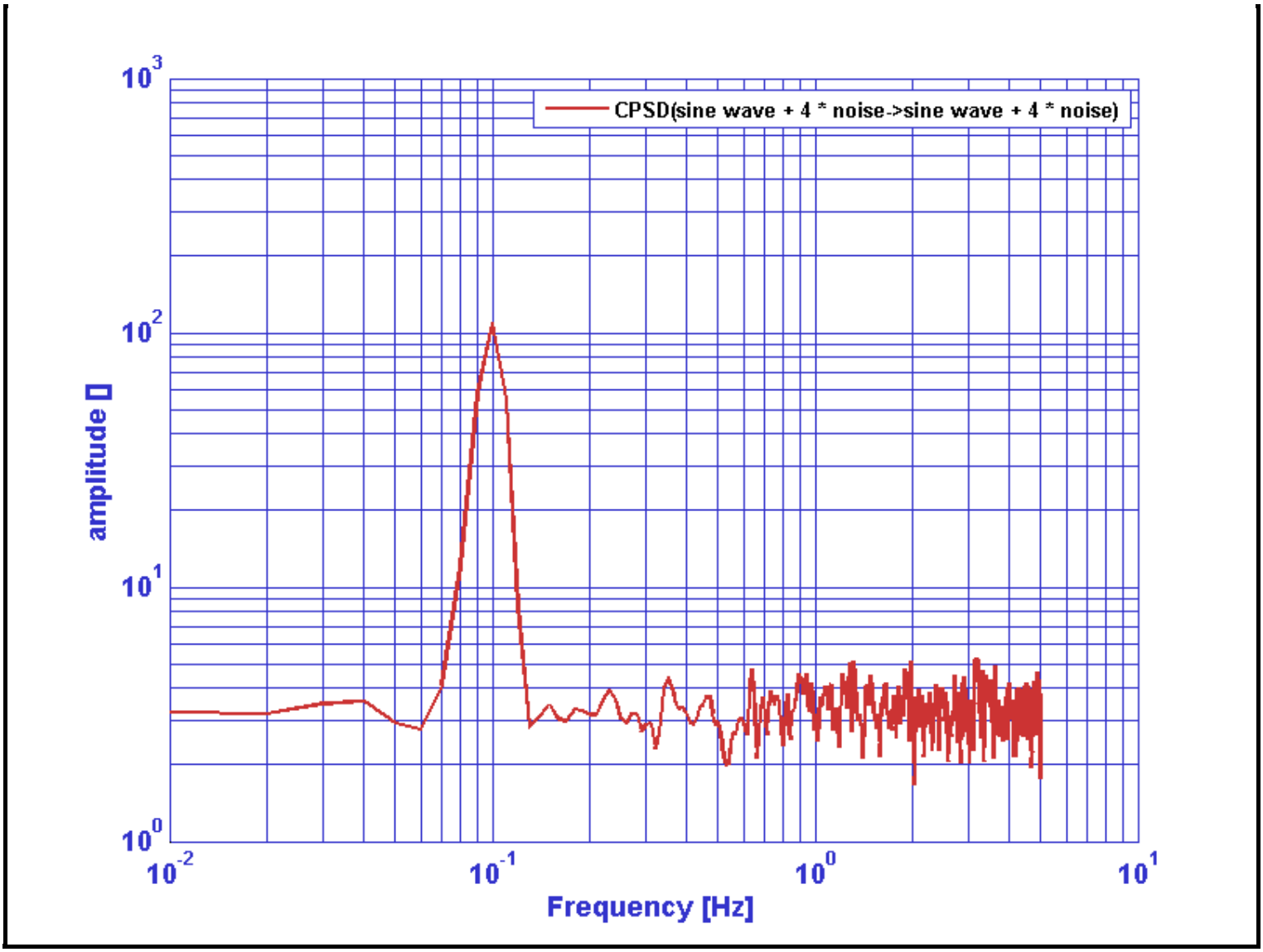
Example

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));
```


CPSD(sine wave + noise->sine wave + noise)
amplitude []
Frequency [Hz]
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-2}
10^{-1}
10^{0}
10^{1}


CPSD(sine wave + noise->sine wave + 4 * noise)
amplitude []
Frequency [Hz]
Phase [deg]
Frequency [Hz]



Power spectral density estimates Cross coherence estimates

©LTP Team

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

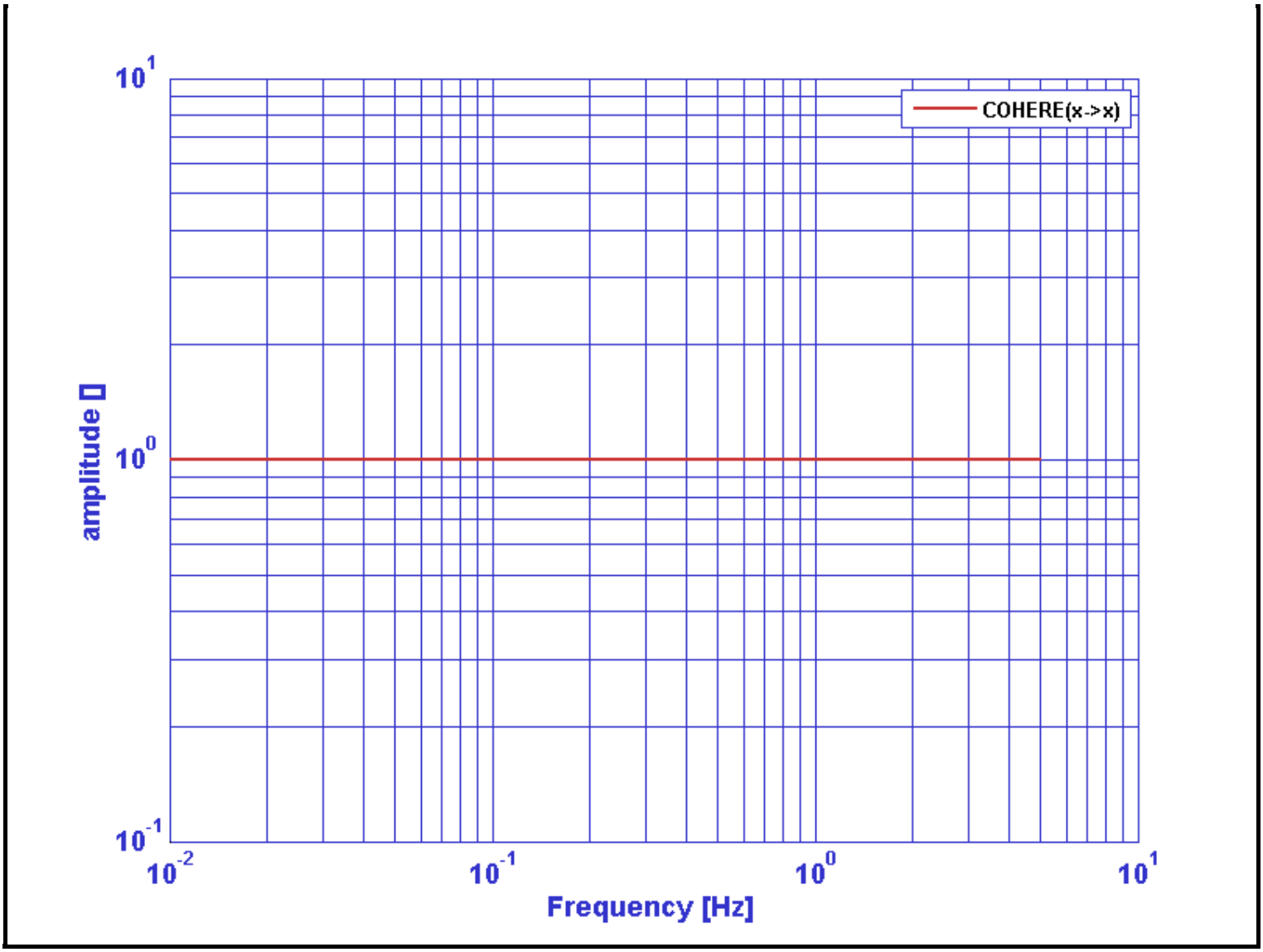
The length of the window is set by the value of the parameter `'Nfft'`, so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

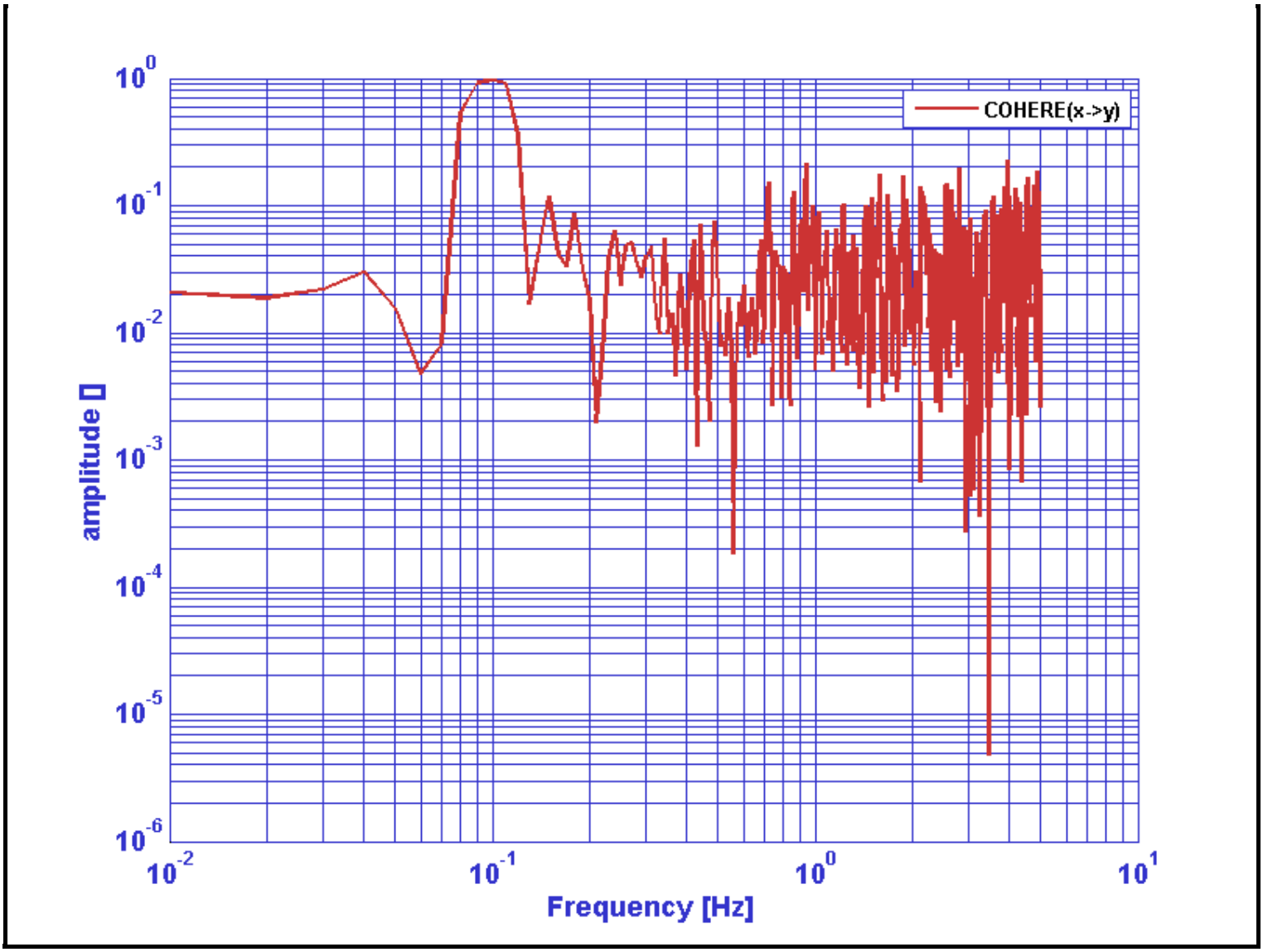
```
  If the user doesn't specify the value of a given parameter, the default value is used.
  The function makes coherence estimates between all input AOs. Therefore, if the input argument
list contains N analysis objects, the output a will contain NxN coherence estimates.
The diagonal elements will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


COHERE(x->x)
amplitude []
Frequency [Hz]
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-1}
10^{0}
10^{1}



Cross-spectral density estimates Transfer function estimates

©LTP Team

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 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_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

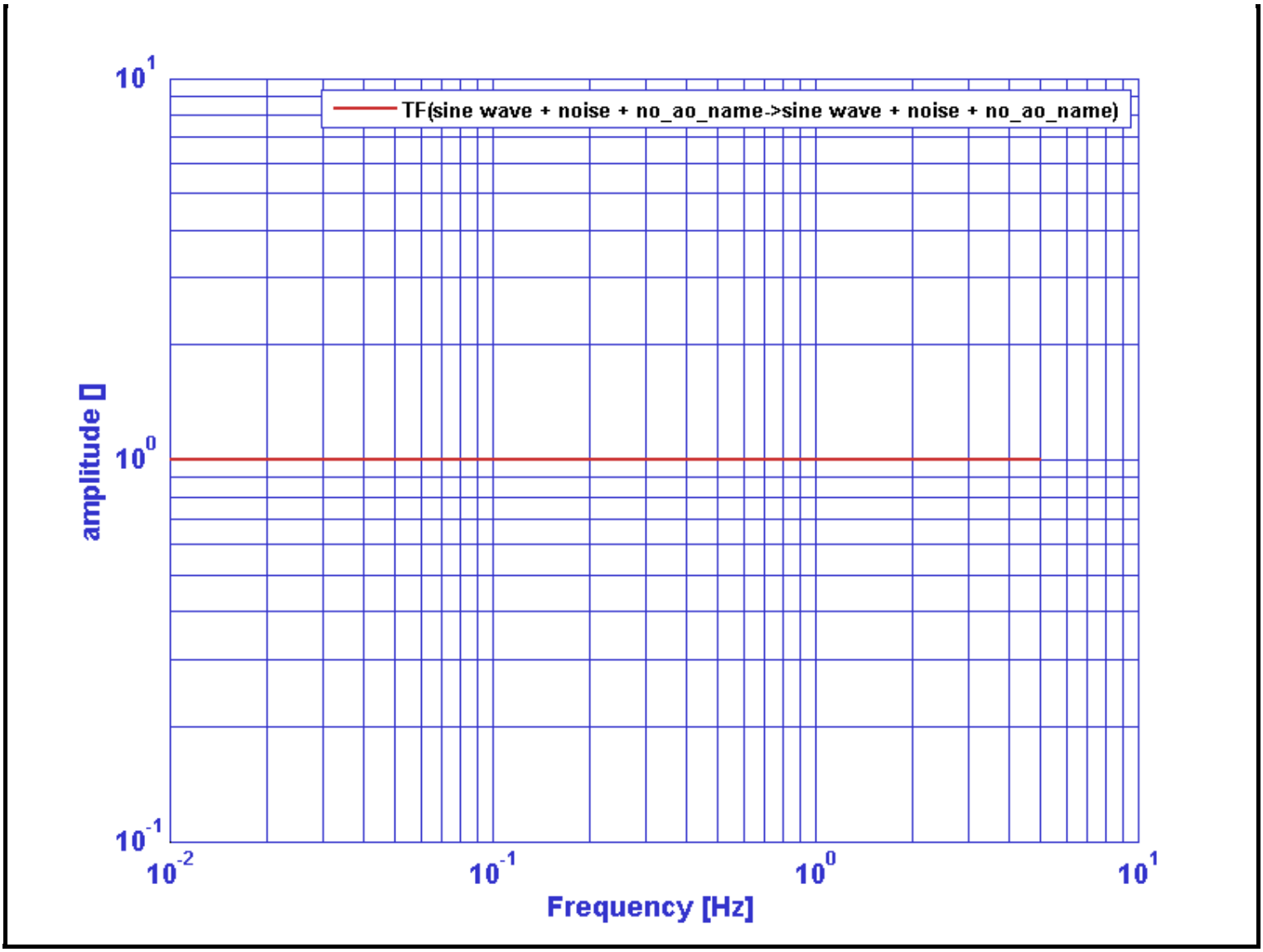
The length of the window is set by the value of the parameter `'Nfft'`, so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

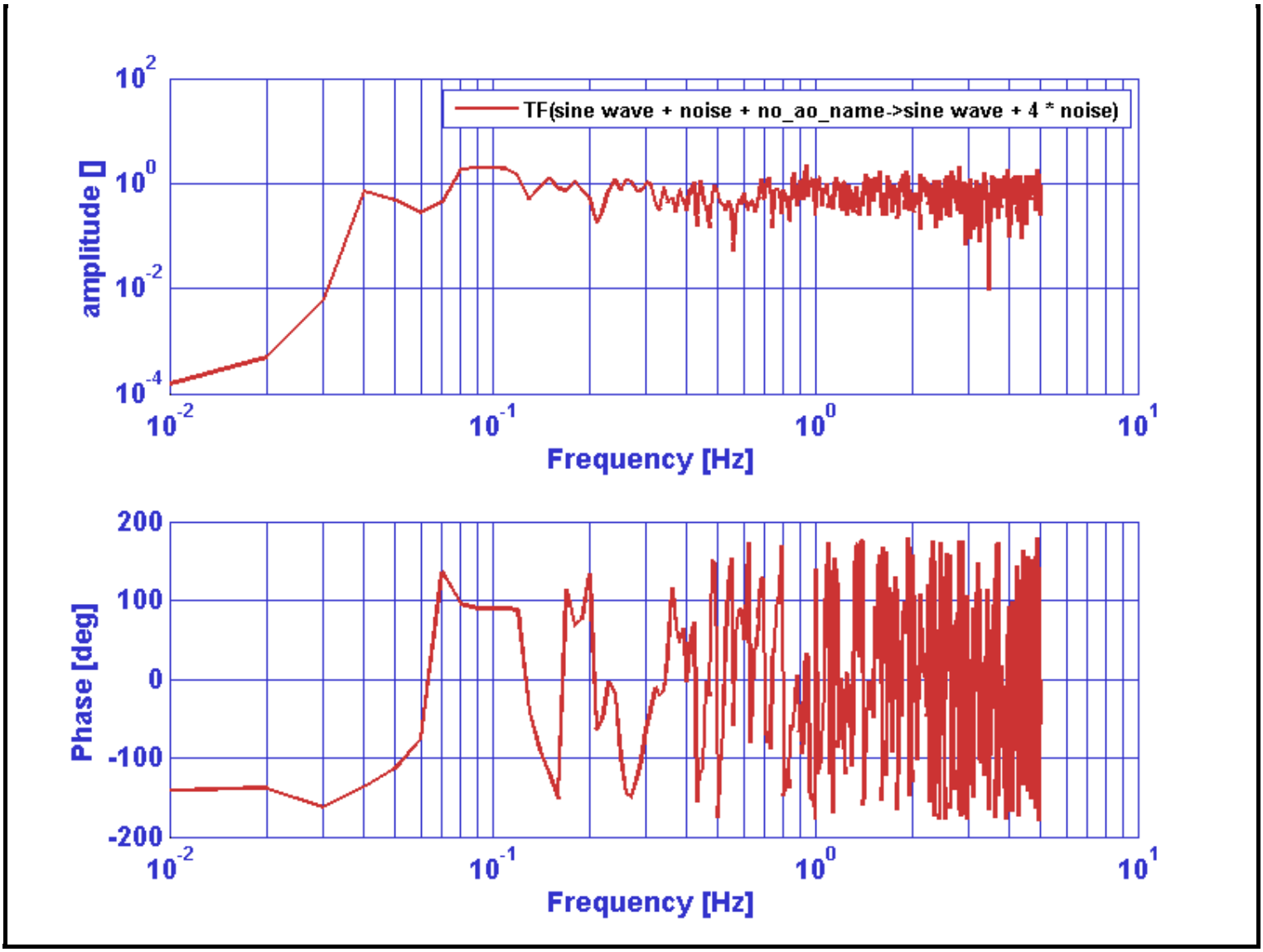
```
  If the user doesn't specify the value of a given parameter, the default value is used.
  The function makes transfer functions estimatesbetween all input AOs. Therefore, if the input
argument
list contains N analysis objects, the output a will contain NxN TFE estimates.
The diagonal elements will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

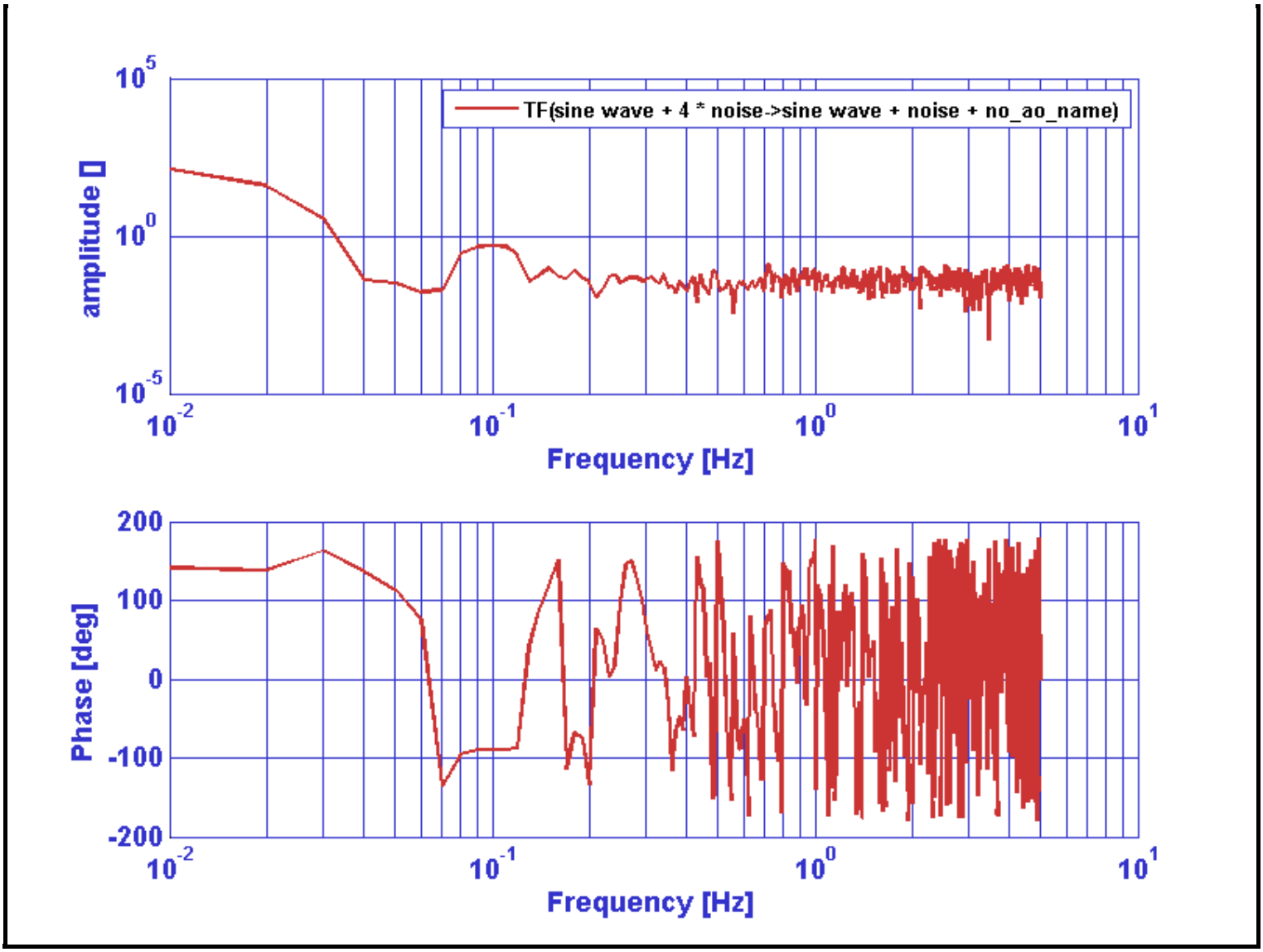
Example

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));
```


TF(sine wave + noise + no_ao_name->sine wave + noise + no_ao_name)
amplitude []
Frequency [Hz]
10^1
10^0
10^-1
10^-2
10^-1
10^0
10^1


TF(sine wave + noise + no_ao_name->sine wave + 4 * noise)
amplitude []
Frequency [Hz]
Phase [deg]
Frequency [Hz]



Cross coherence estimates
Log-scale power spectral density estimates

©LTP Team

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 freqencies which are linear multiples of the minimum frequency resolution `1/T` where `T` is the window lenght, 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 polinomial 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 freqency 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.

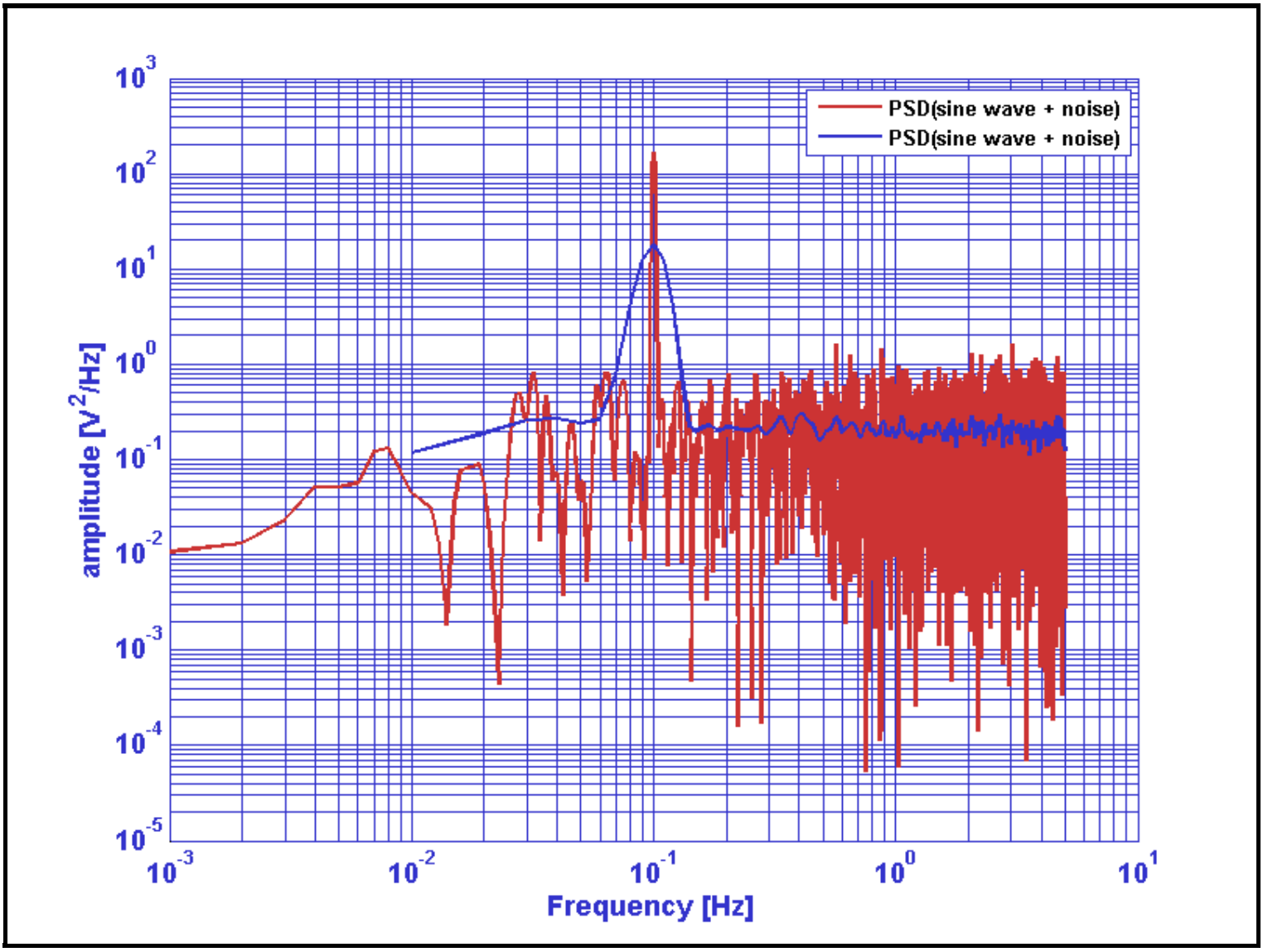
```
If the user doesn't specify the value of a given parameter, the default value is used.
```

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);
    iplot(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 freqencies which are linear multiples of the minimum frequency resolution `1/T` where `T` is the window lenght, 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 polinomial 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`, ... 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:

- `'Kdes'` - the desired number of averages [default: 100]
- `'Jdes'` - the number of spectral frequencies to compute [default: fs/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

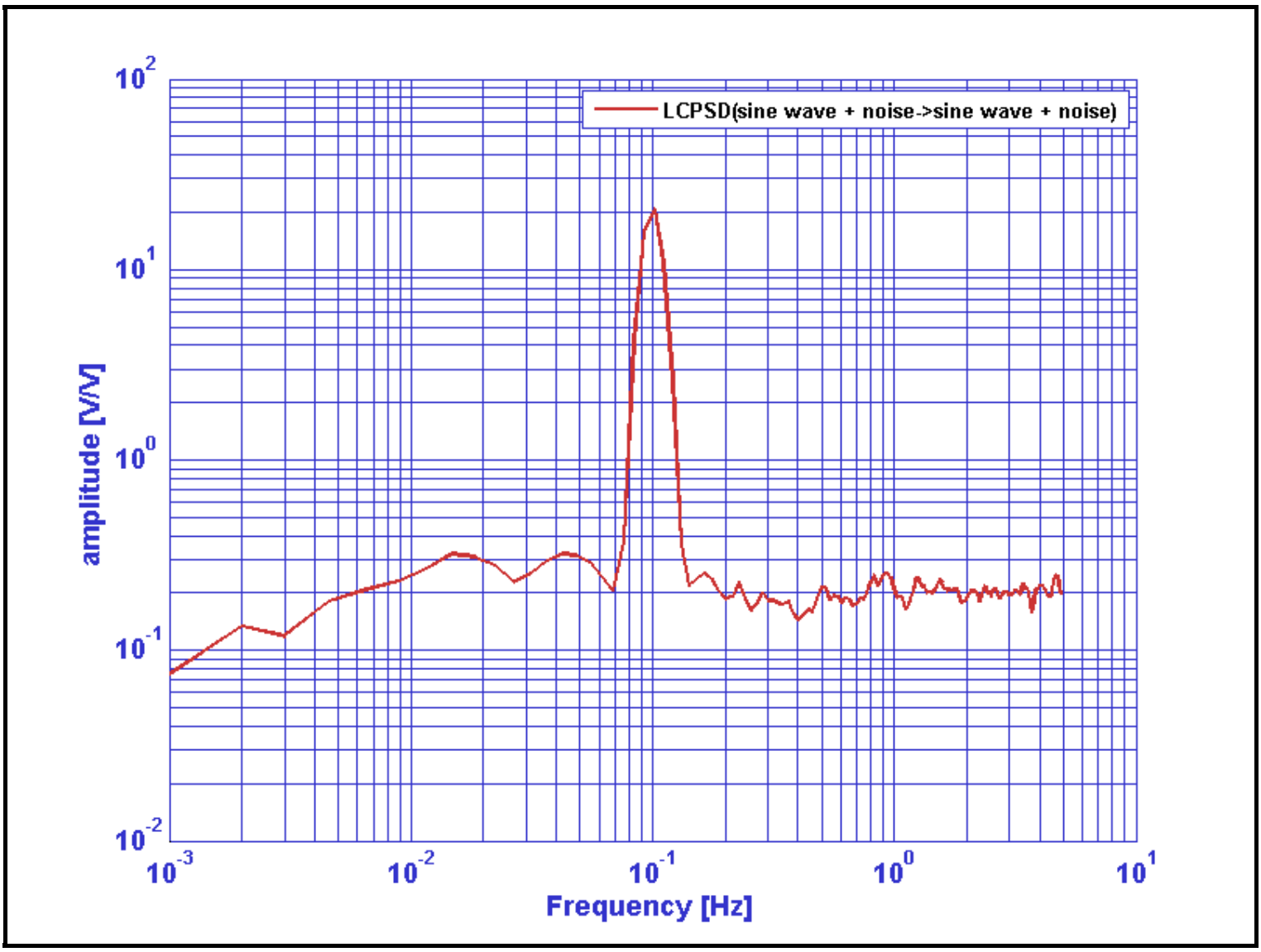
The length of the window is recalculated for each freqency bin, so the 'Win' parameter is used only to provide the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

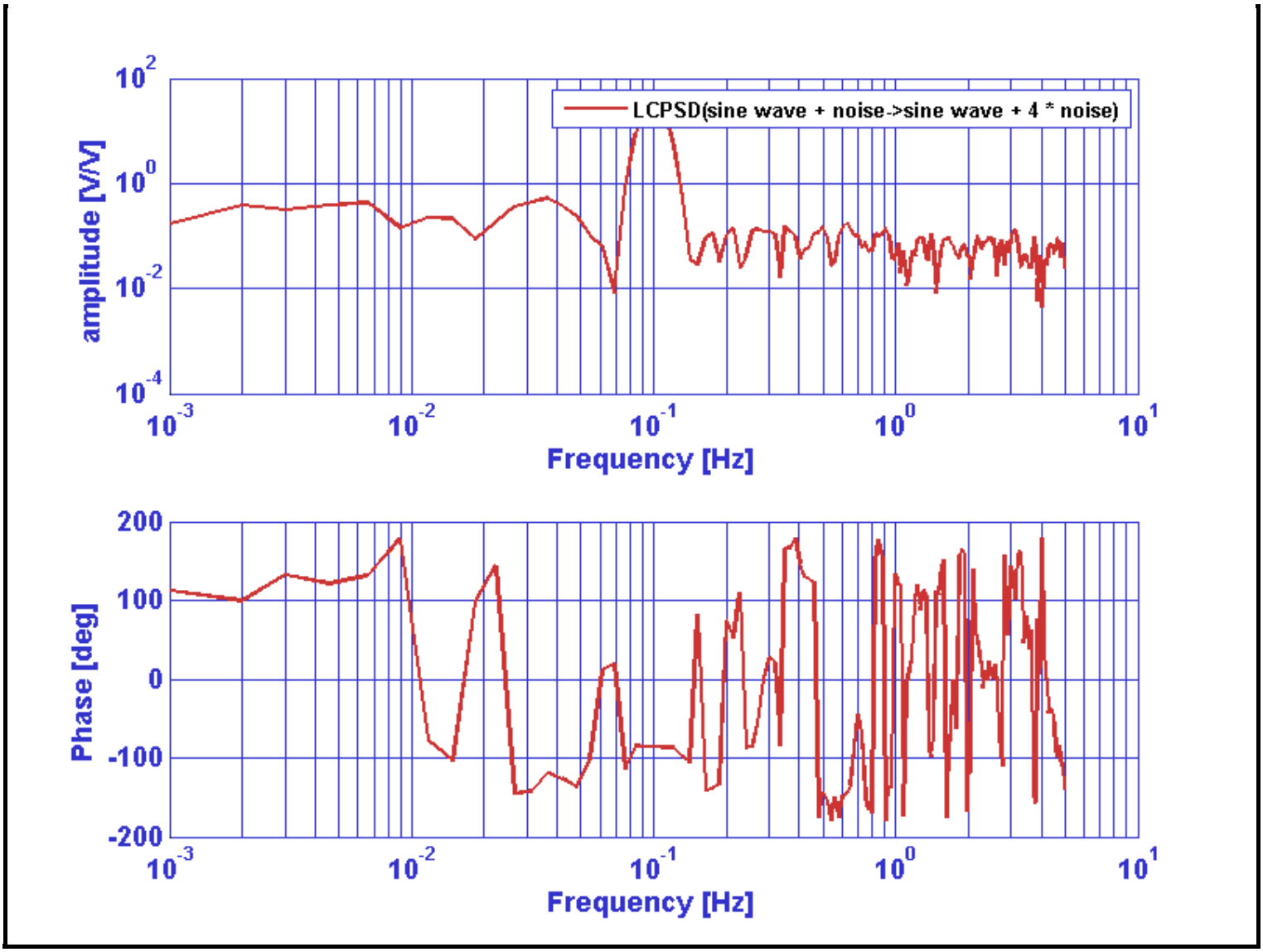
```
  If the user doesn't specify the value of a given parameter, the default value is used.
  The function makes CPSD estimates between all input AOs. Therefore, if the input argument
list contains N analysis objects, the output a will contain NxN CPSD estimates.
The diagonal elements will be S_ai_ai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

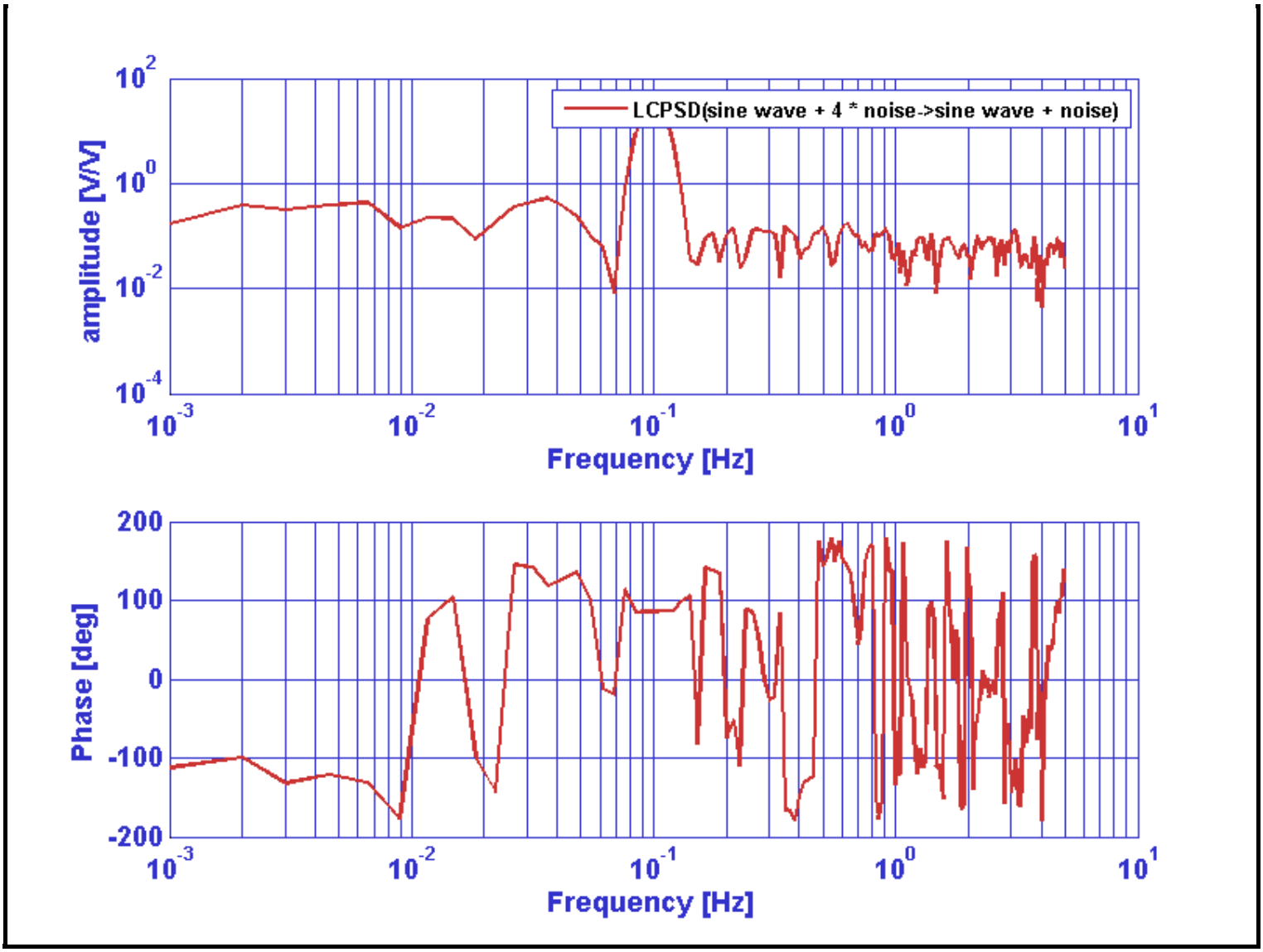
Example

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));
```


LCPSD(sine wave + noise->sine wave + noise)
amplitude [V/V]
Frequency [Hz]
10^2
10^1
10^0
10^-1
10^-2
10^-3
10^-2
10^-1
10^0
10^1


LCPSD(sine wave + noise->sine wave + 4 * noise)
amplitude [V/V]
Frequency [Hz]
Phase [deg]
Frequency [Hz]



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 freqencies which are linear multiples of the minimum frequency resolution `1/T` where `T` is the window lenght, 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 polinomial 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 >= 2. `b` includes the NXN 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

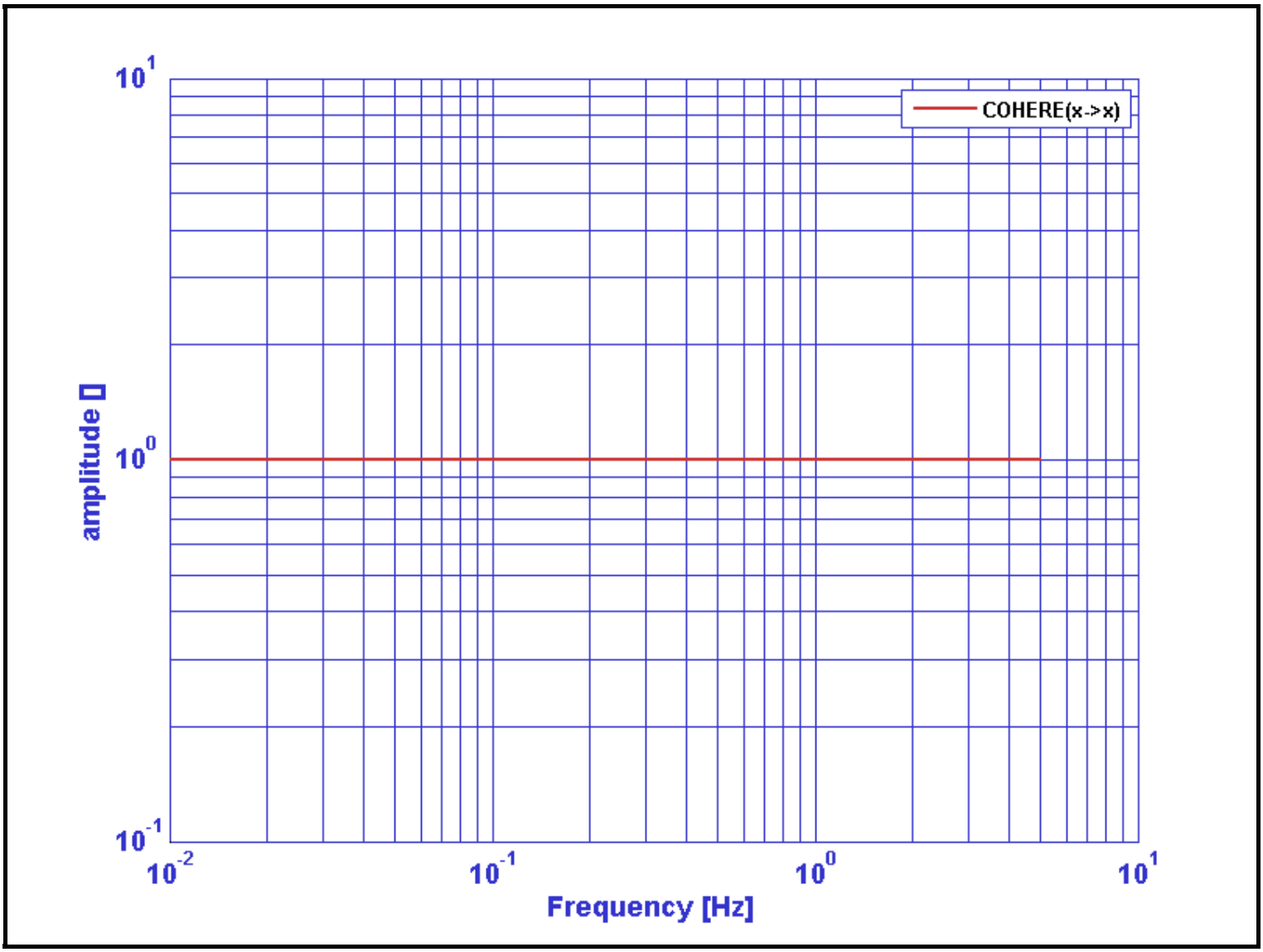
The length of the window is recalculated for each freqency bin, so the 'Win' parameter is used only to provide the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

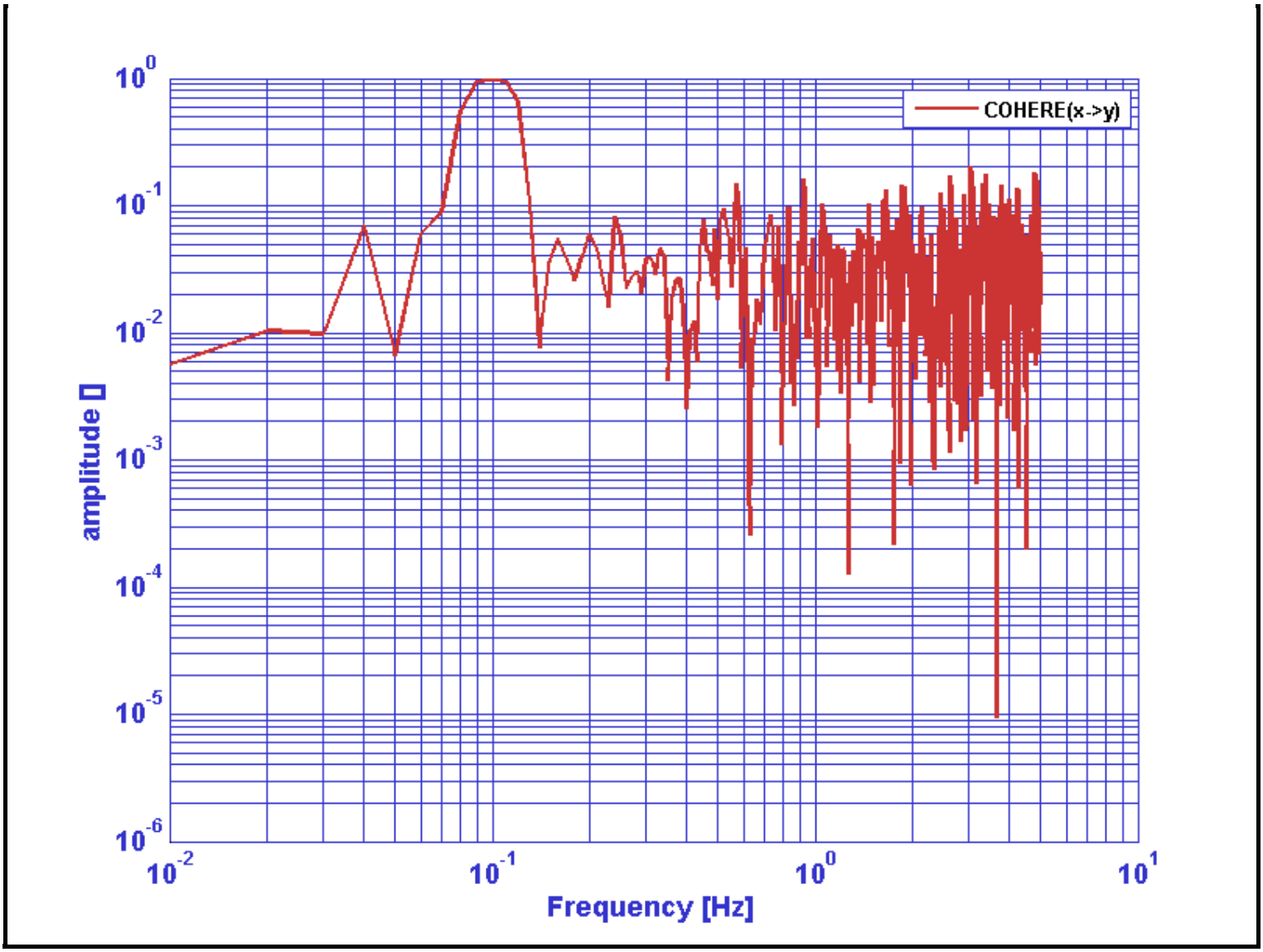
```
  If the user doesn't specify the value of a given parameter, the default value is used.
  The function makes coherence estimates between all input AOs. Therefore, if the input argument
list contains N analysis objects, the output a will contain NxN coherence estimates.
The diagonal elements will 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')));
    z = ltpda_lcohere(x,y,pl);
    iplot(z(1,1));
    iplot(z(1,2));
```


COHERE(x->x)
amplitude []
10^1
10^0
10^-1
10^-2
10^-1
10^0
10^1
Frequency [Hz]



Log-scale cross-spectral density estimates

Log-scale transfer function estimates

©LTP Team

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 freqencies which are linear multiples of the minimum frequency resolution `1/T` where `T` is the window lenght, 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_ltfe` 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 polinomial of time in order to reduce the impact of the border discontinuities. Detrending is performed on each individual window.

Syntaxis

```
        b = ltpda_ltfe(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:

- `'Kdes'` - the desired number of averages [default: 100]
- `'Jdes'` - the number of spectral frequencies to compute [default: fs/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

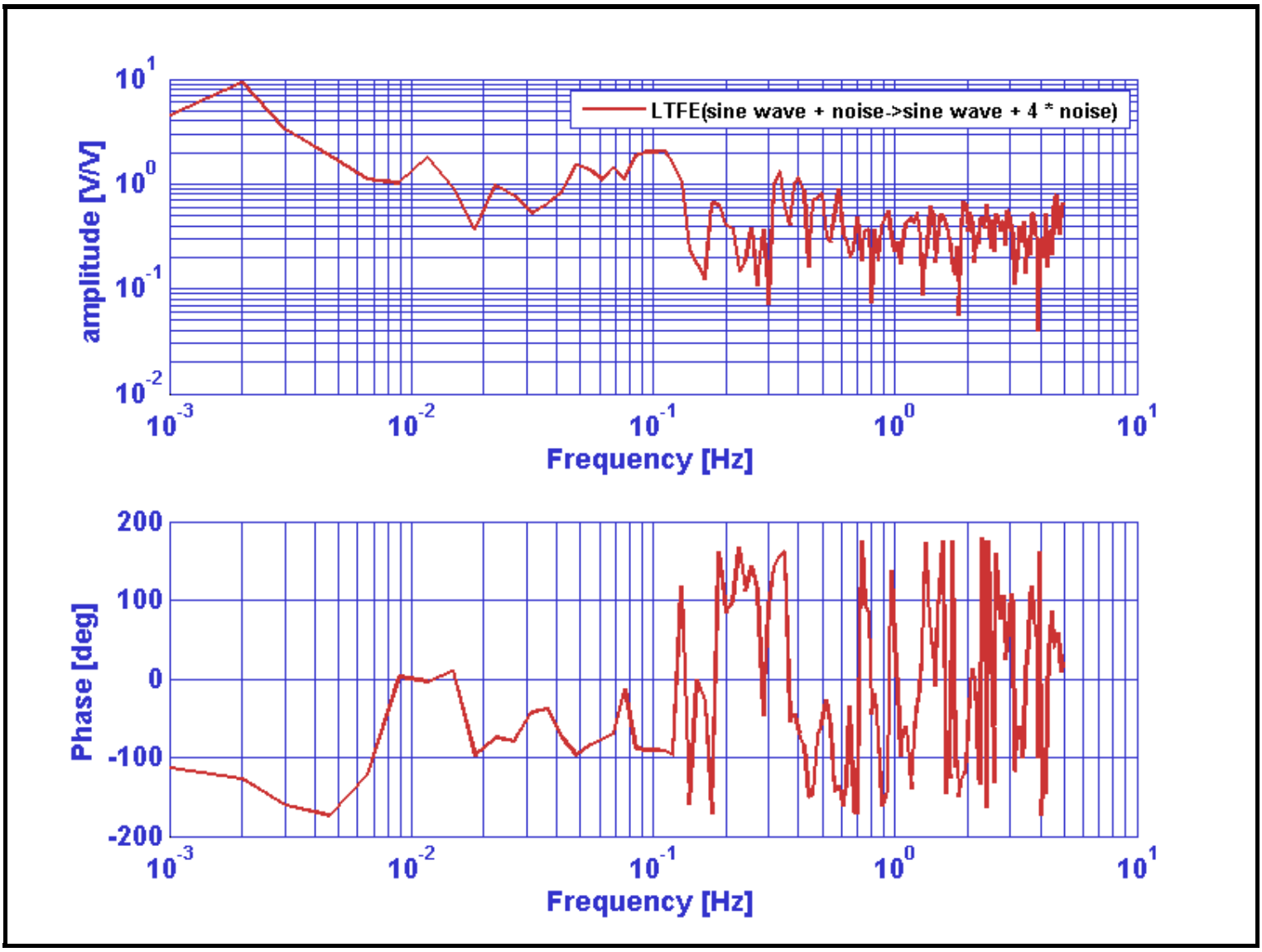
The length of the window is set by the value of the parameter `'Nfft'`, so that the window is actually rebuilt using only the key features of the window, i.e. the name and, for Keiser windows, the PSLL.

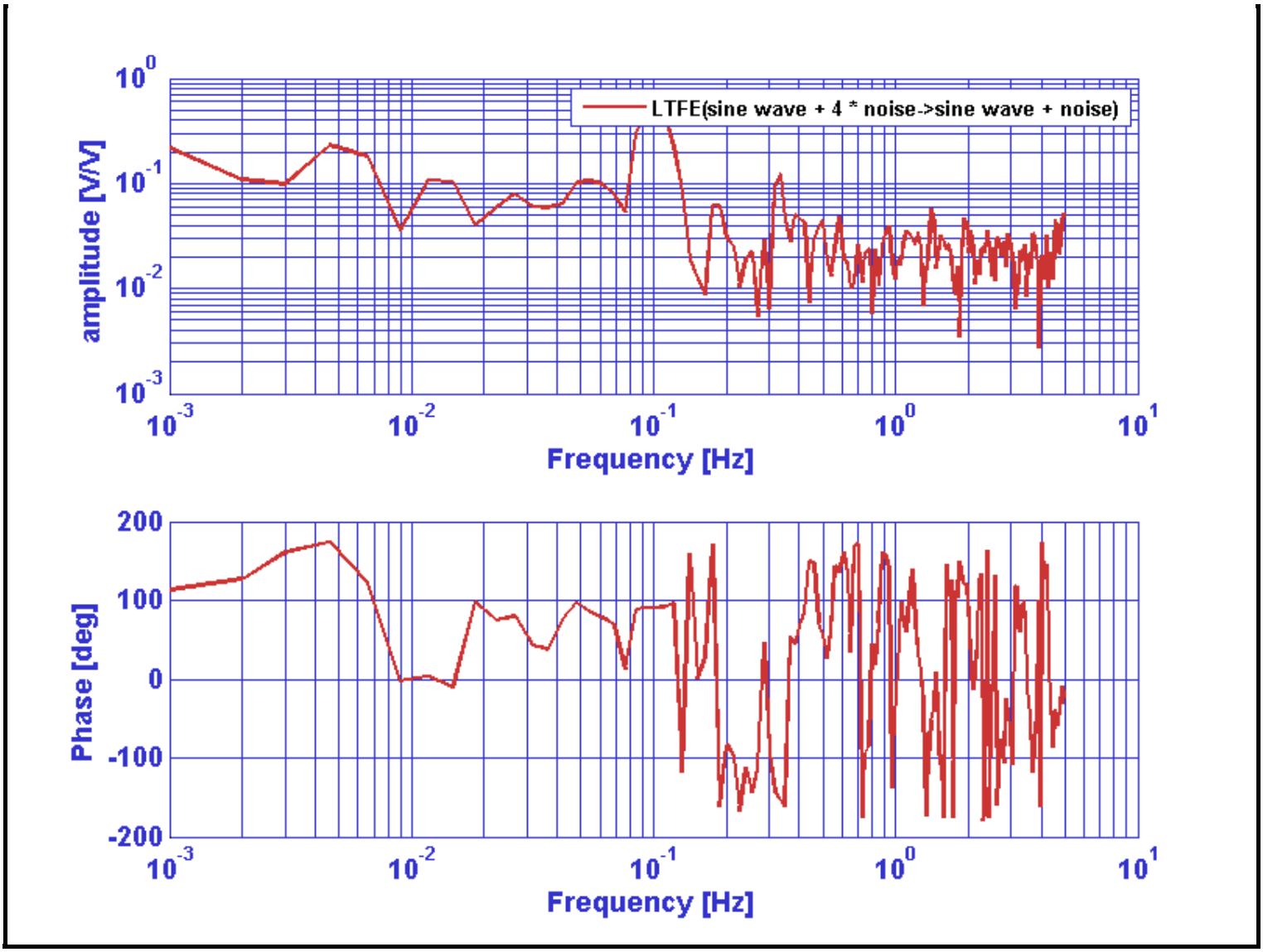
```
  If the user doesn't specify the value of a given parameter, the default value is used.
  The function makes transfer functions estimatesbetween all input AOs. Therefore, if the input
argument
list contains N analysis objects, the output a will contain NxN TFE estimates.
The diagonal elements will 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_ltfe(x,y,plist('win',specwin(plist('name','BH92'))));
    iplot(z(1,2));
    iplot(z(2,1));
```


LTFE(sine wave + noise->sine wave + 4 * noise)
amplitude [V/V]
Frequency [Hz]
Phase [deg]
Frequency [Hz]



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

Log-scale transfer function estimates Polynomial Fitting

©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.

Fitting Algorithms Time domain Fit

©LTP Team

Time domain Fit

---

ltpda_timedomainfit.m uses the MATLAB function `lscov.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:

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

`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 `coeffsAO` 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.

©LTP Team

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

Time domain Fit The LTPDA Launch Bay

©LTP Team

The LTPDA Launch Bay

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

```
>> ltpdalauncher
```

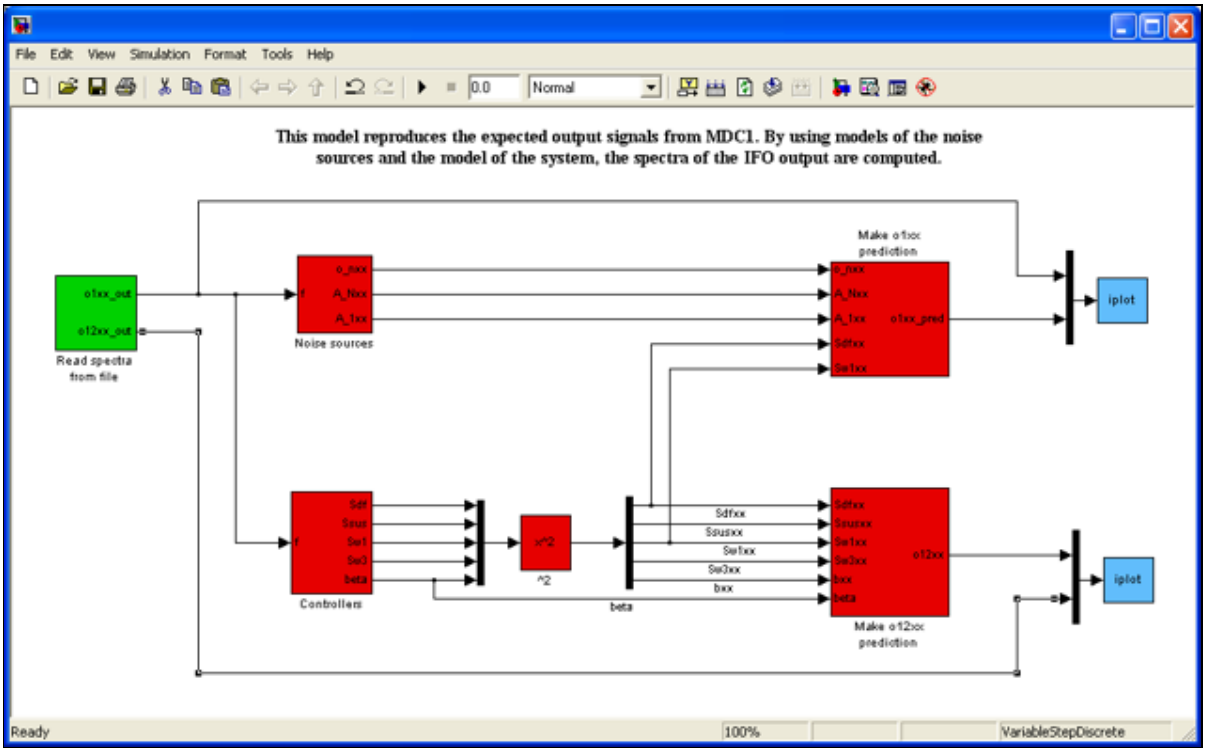
The Launch Bay is automatically started from the `ltpda_startup` script.

©LTP Team

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` .

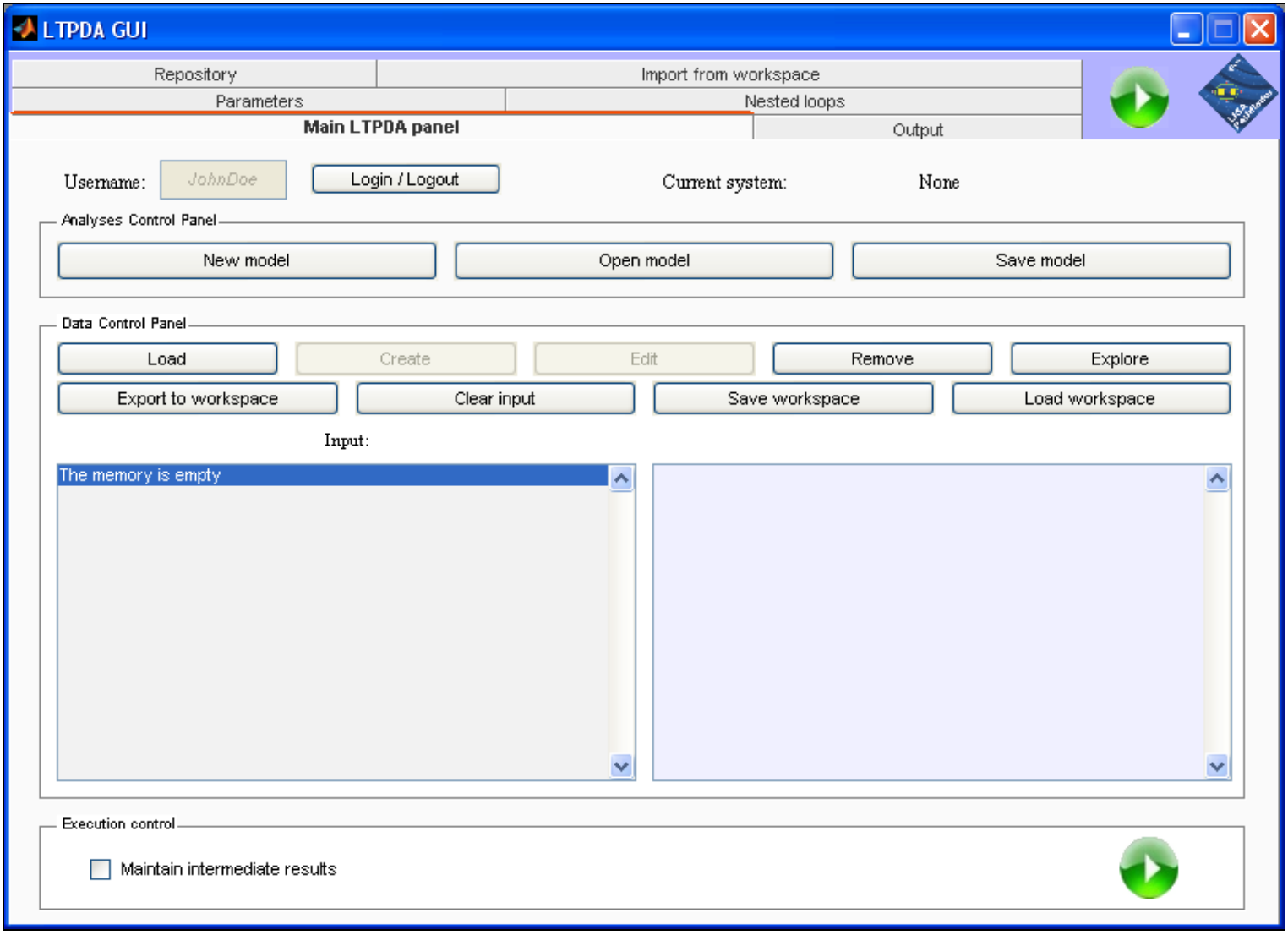
```
>> ltpdagui
```

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.

The LTPDA Launch Bay Basics

©LTP Team

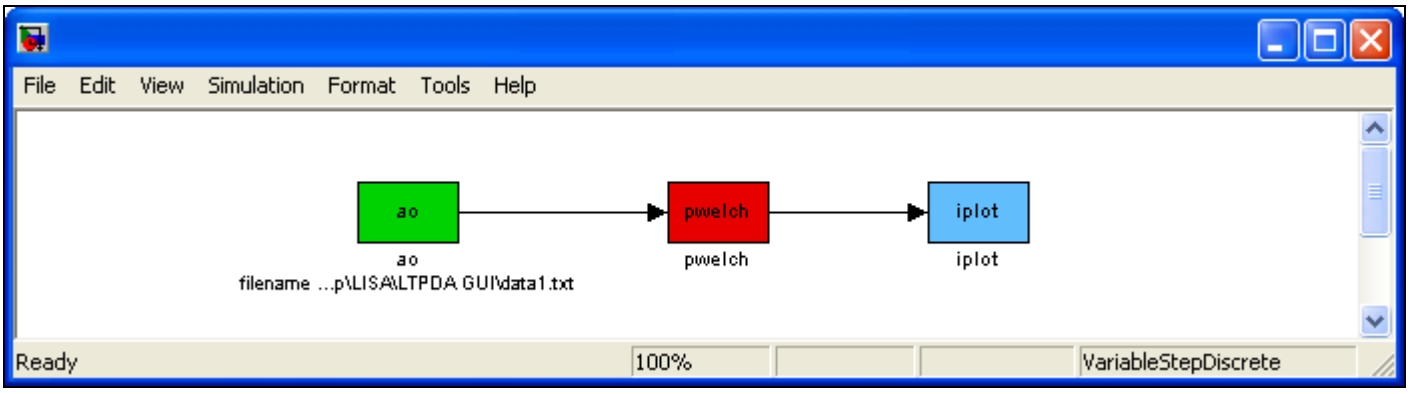
Basics

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 analisys 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.

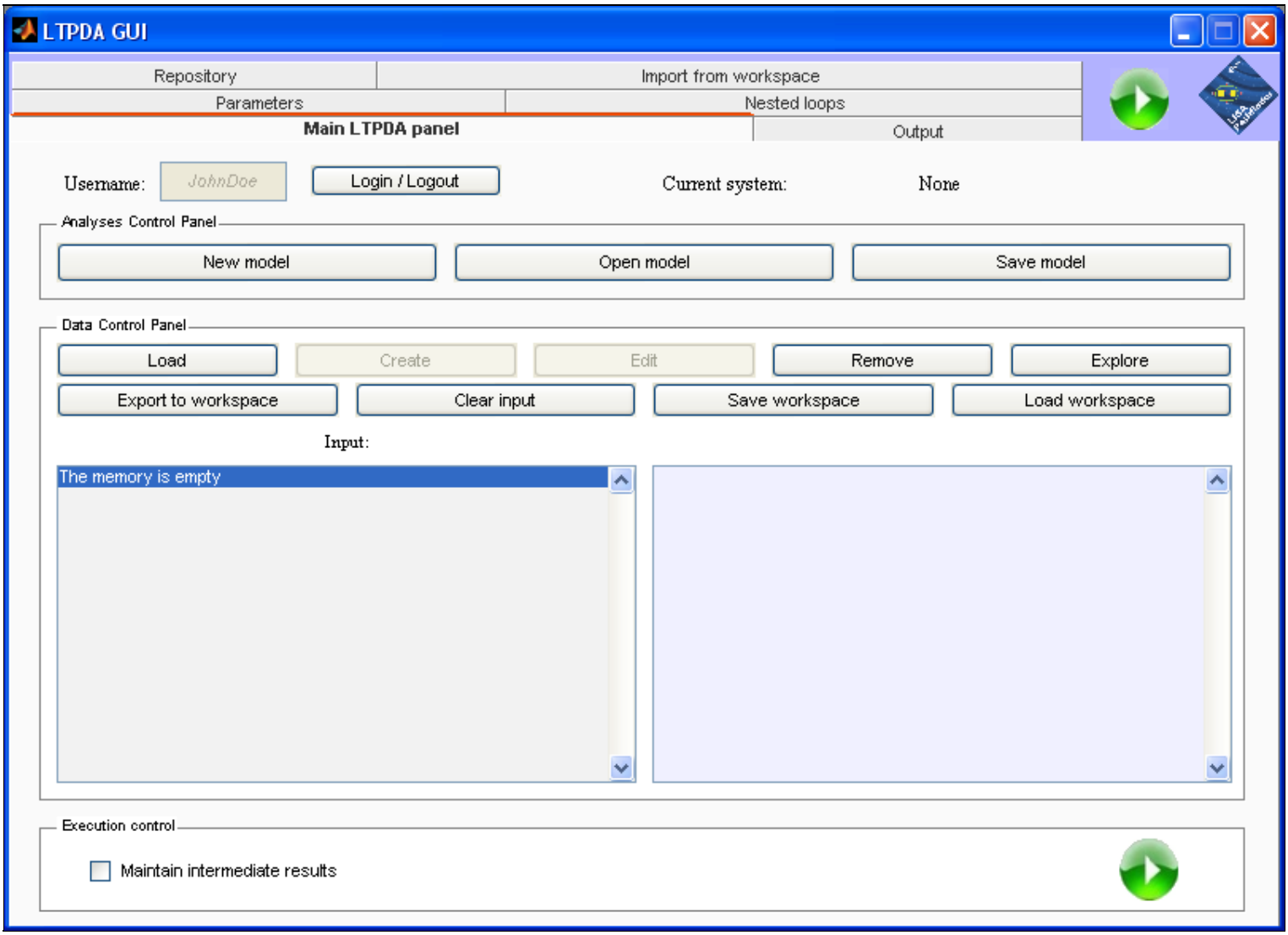
For more information on parameters handling, see the paragraph regarding the Parameters panel.

The LTPDA Analysis GUI Main panel

©LTP Team

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

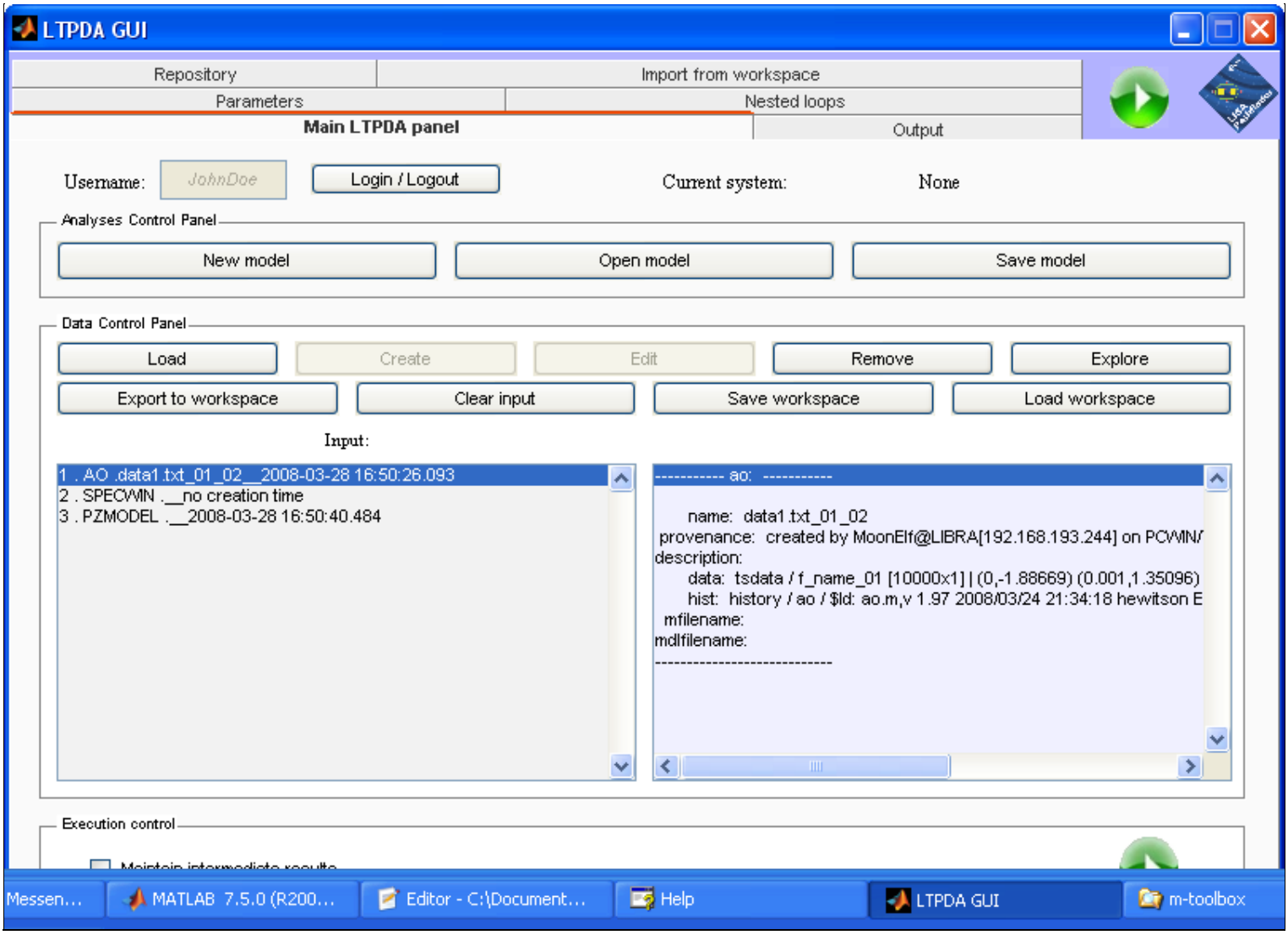
This contains 3 buttons, all devoted to the control of LTPDA Simulink model files.

- '**`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:

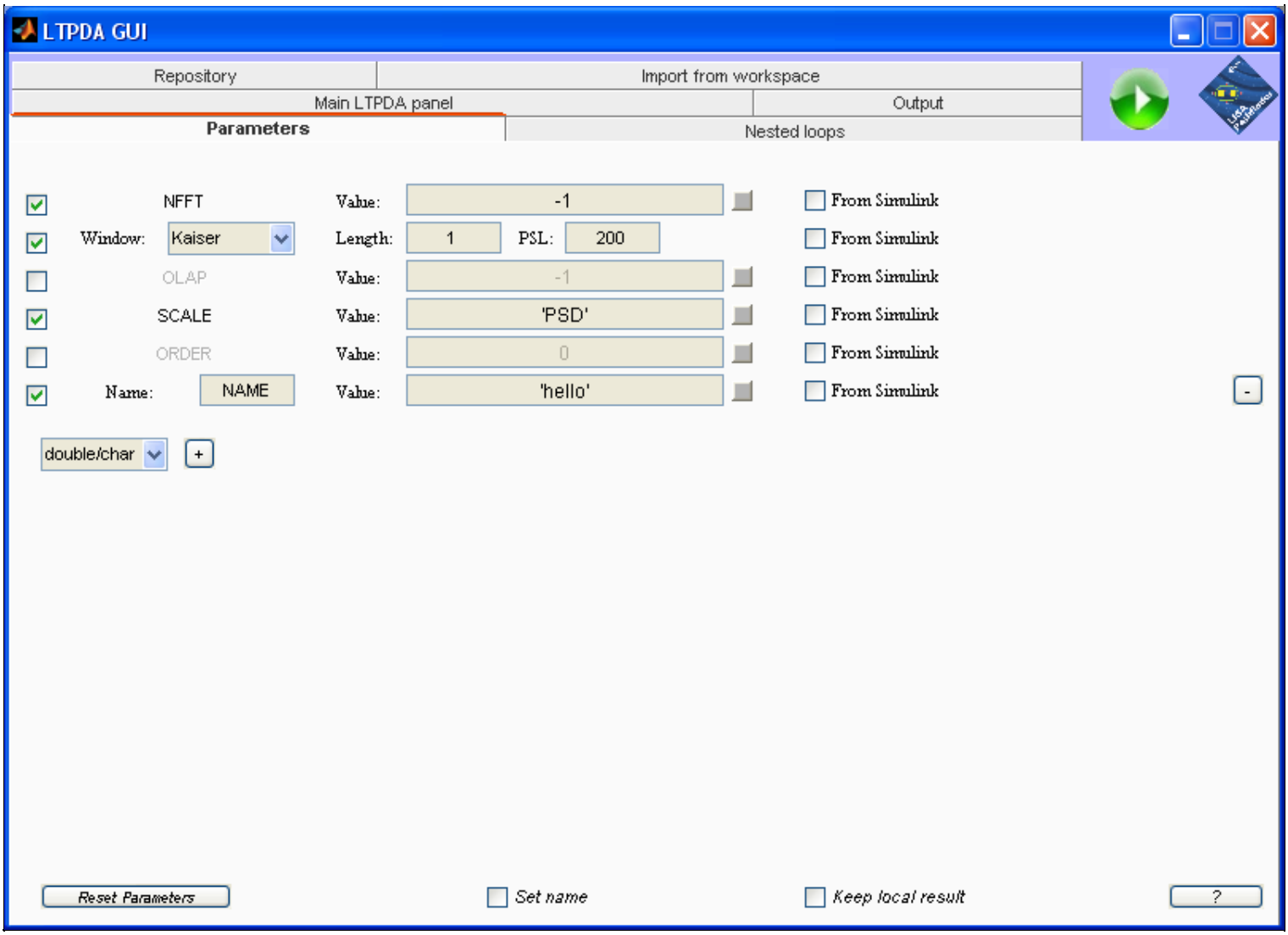
- No block is currently selected.
- The currently selected Simulink model is not a valid LTPDA model.
- The selected block it's not a valid LTPDA block (i.e., it do not contain a proper LTPDA function which answers the call for parameters).

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 someway expected, for example if it 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.

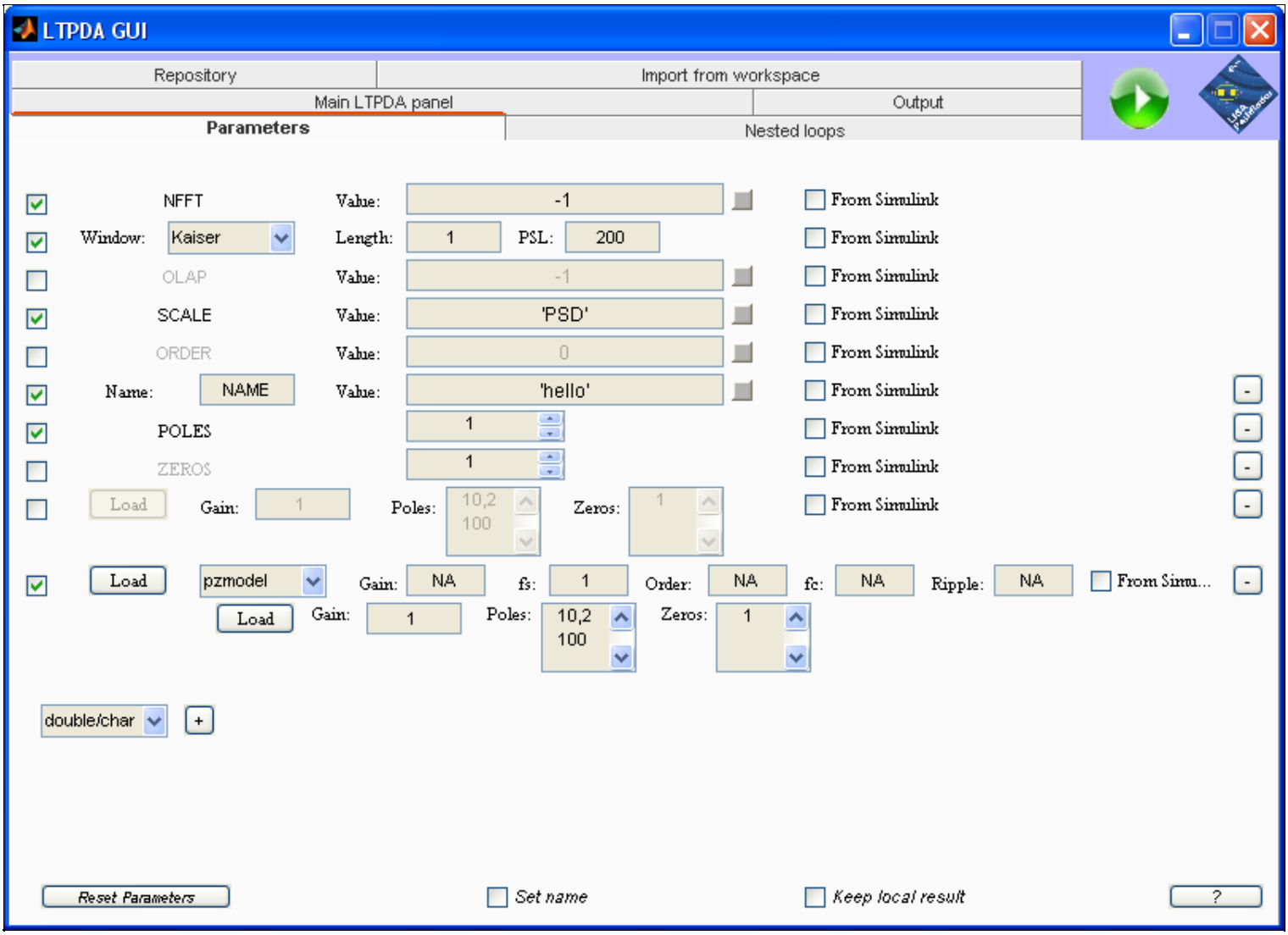
In the parameters panel the difference among required and added parameters is shown on the far right: added parameters have a small '-' button, which lets the user to remove it; required parameters have no such button, since the corresponding parameter was automatically required by the function and not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



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

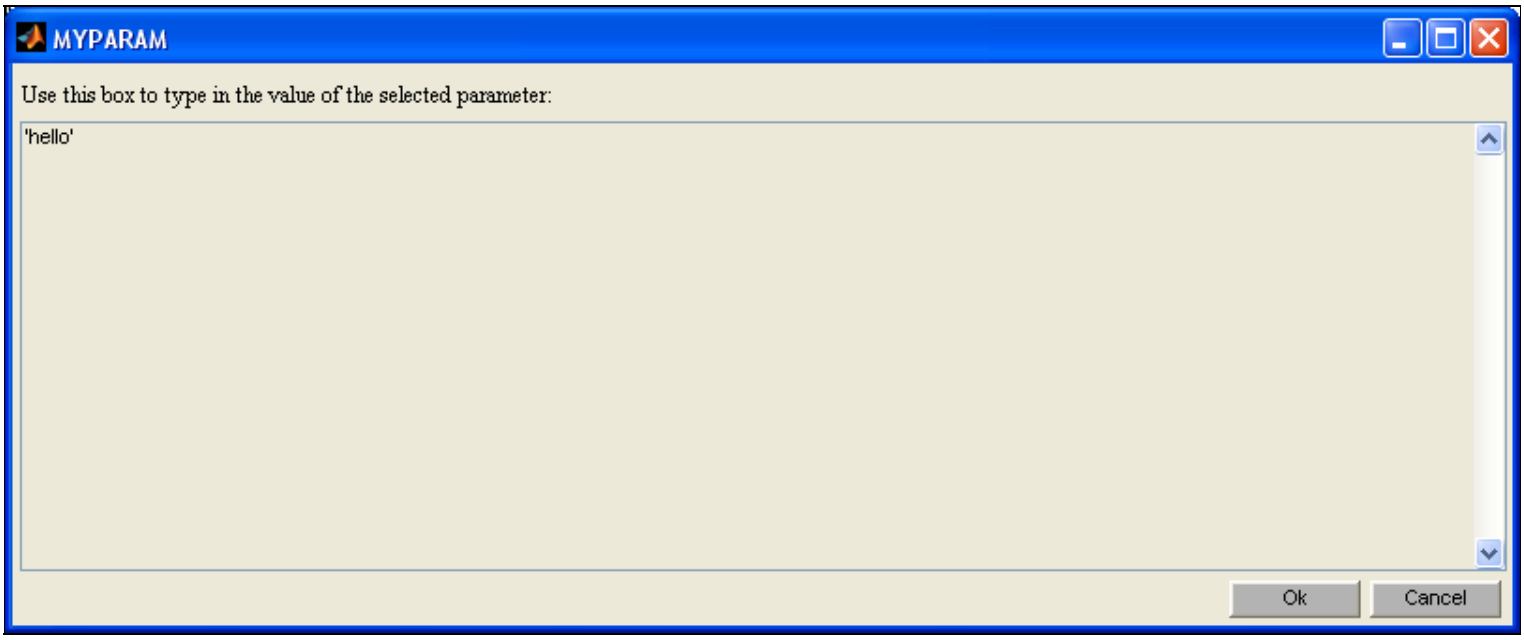
The following field lets the user to type in the parameters value. This is true for every '`Value:`' field in the parameters panel.
Please note that often is possible to have a brief help text in the tooltip string, just moving the mouse pointer over the edit field and waiting a second.
This will explain 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 the output of the evaluated string.

Expand edit field button

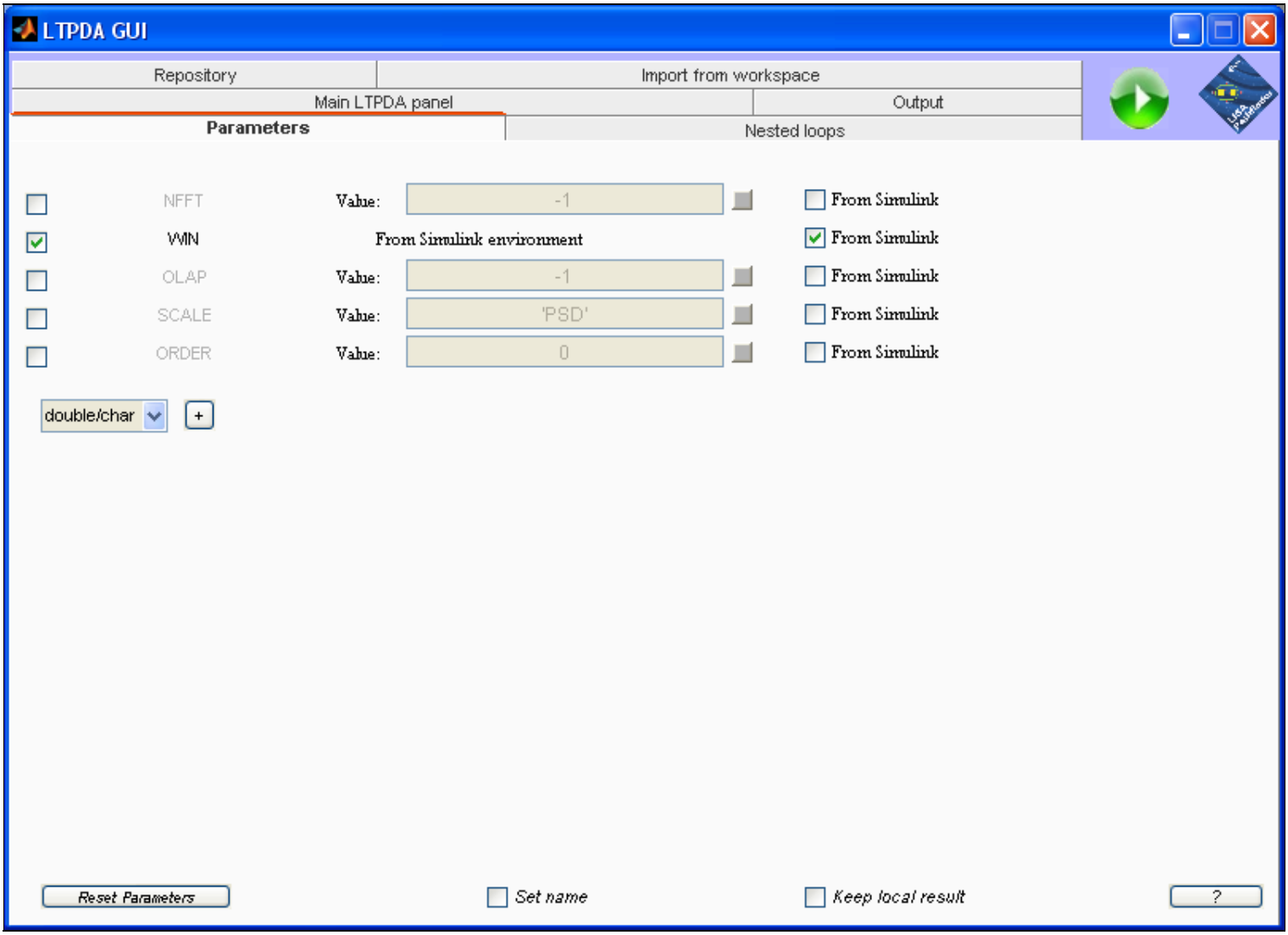
The small gray button aside each parameter's value edit field, for double/char parameters, will open up a window with a bigger edit field, where the user can type in the value more comfortably than in the small one in the panel. This is particularly handy for those case when the parameter's value is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



From Simulink checkbox

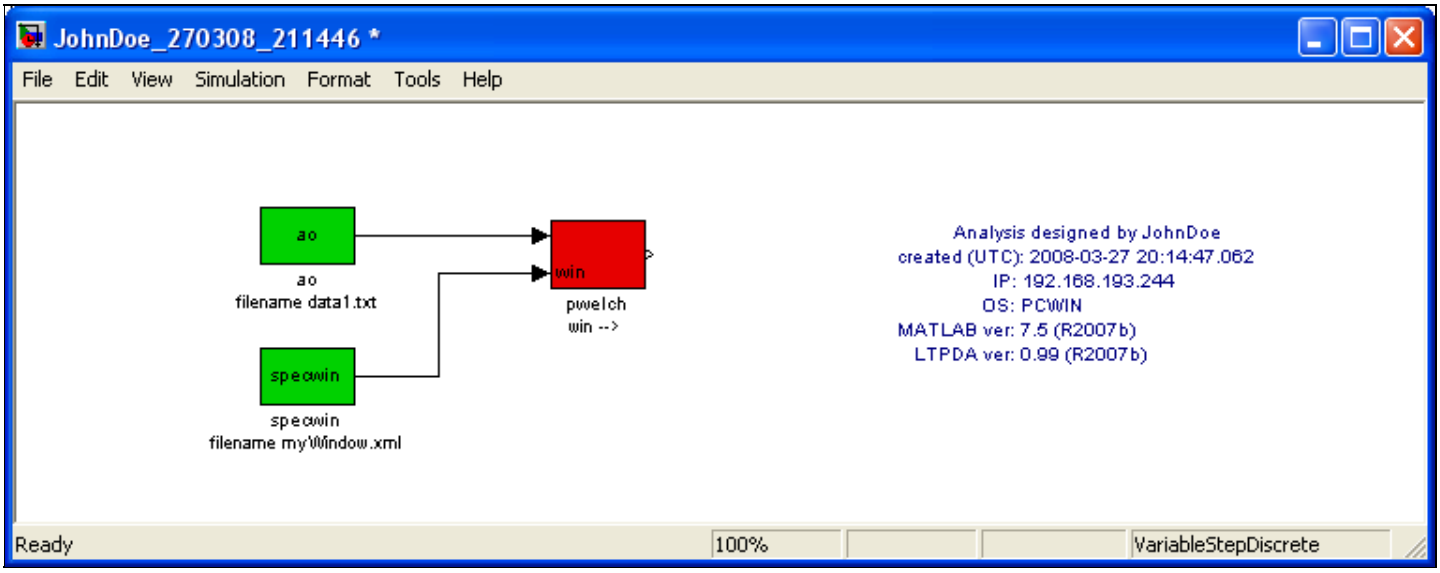
If the parameter is not supposed to be set in the GUI's panel but it must be retrieved from Simulink (for example, using a Constructor block for that particular type of object), the user can click on the

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 inport, 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 '`pwelch`' 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

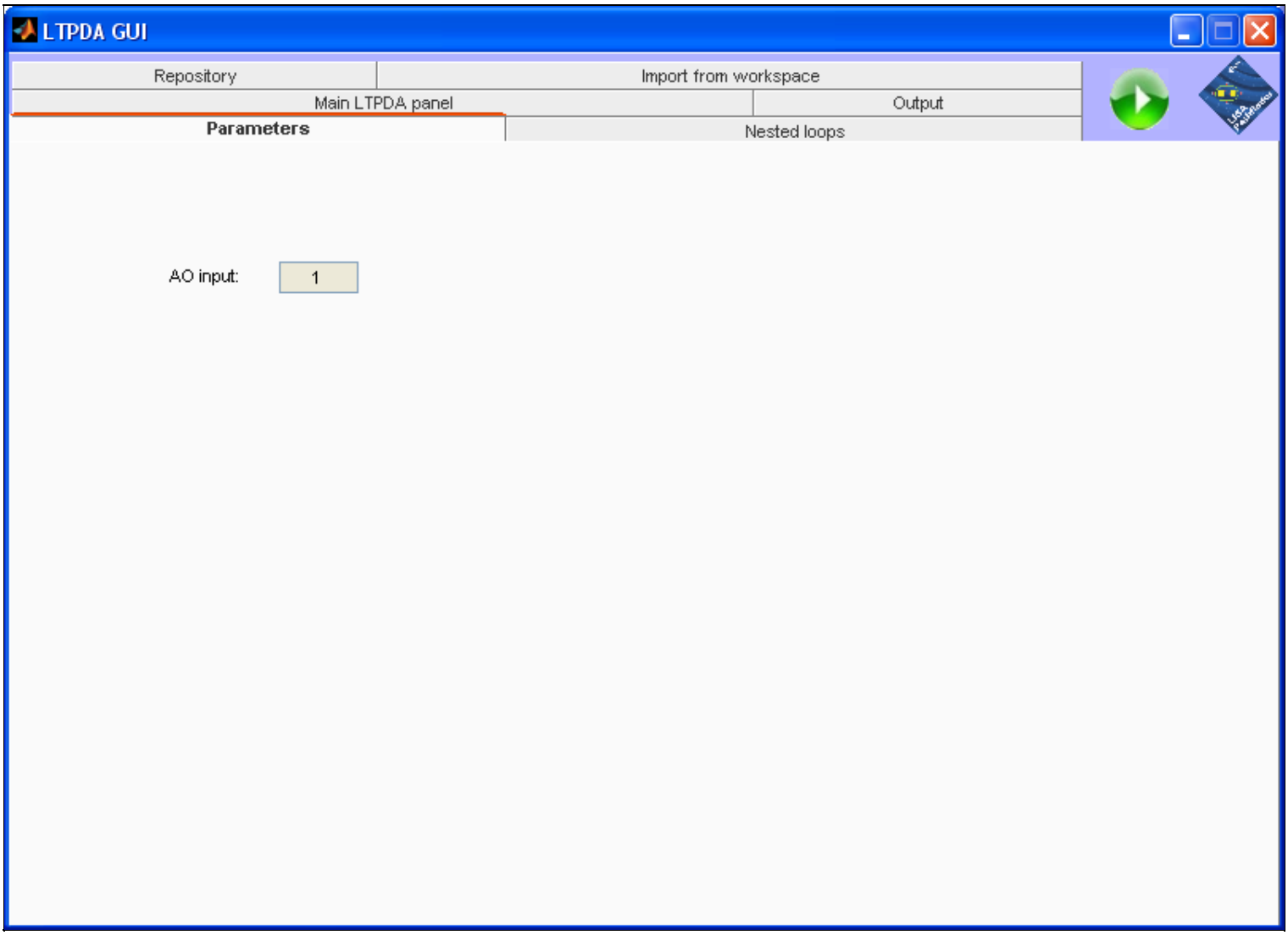
Similarly to the '`Maintain intermediate results` checkbox in the main panel, this checkbox will mark the output of the selected block to be maintained at the end of the analysis, instead of being deleted just like any other intermediate result of the calculation.
In the LTPDA Simulink model the block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

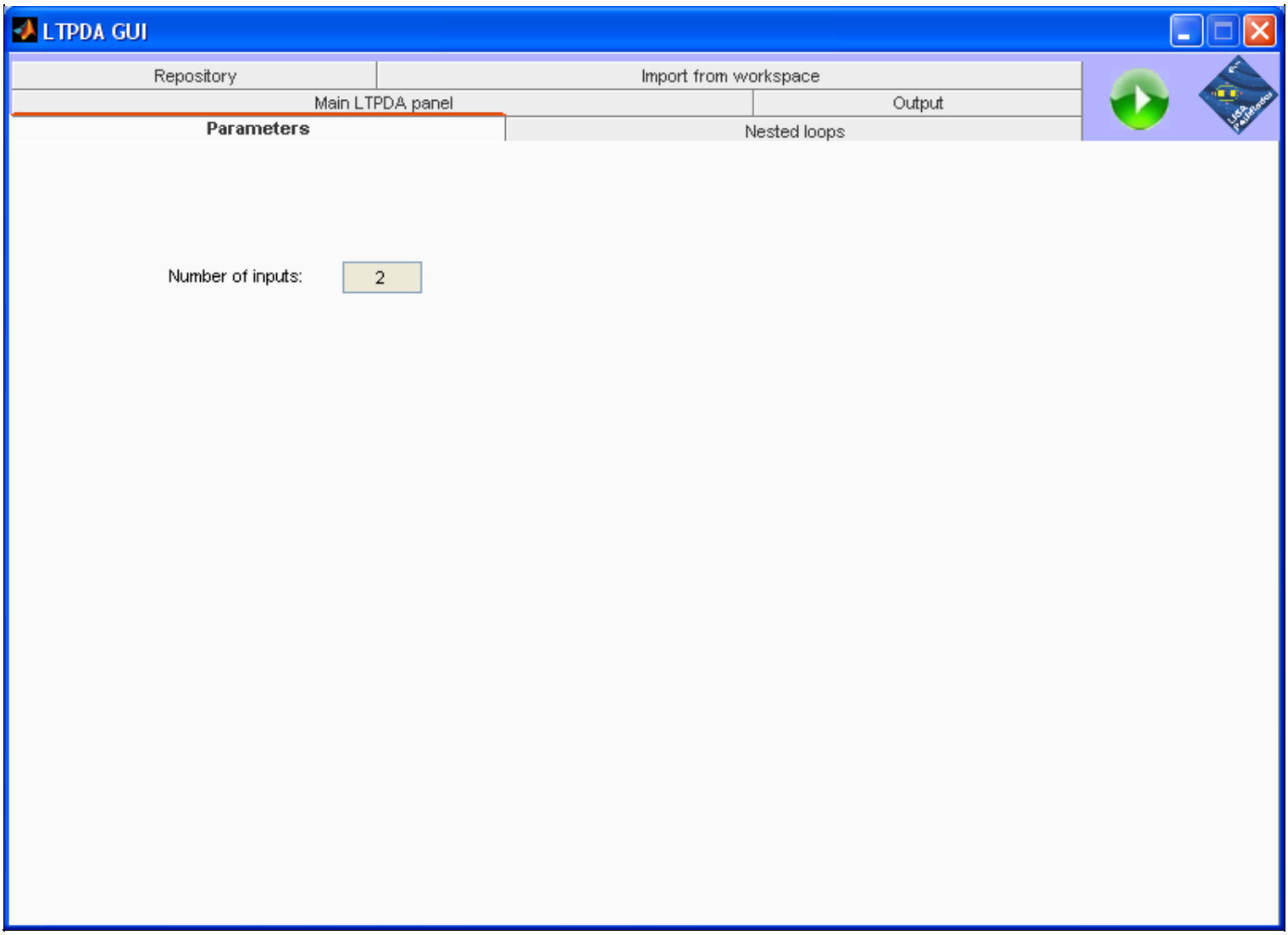
If the user selected a 'Object from list' (input) block into the active LTPDA Simulink model, the GUI will show:



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:

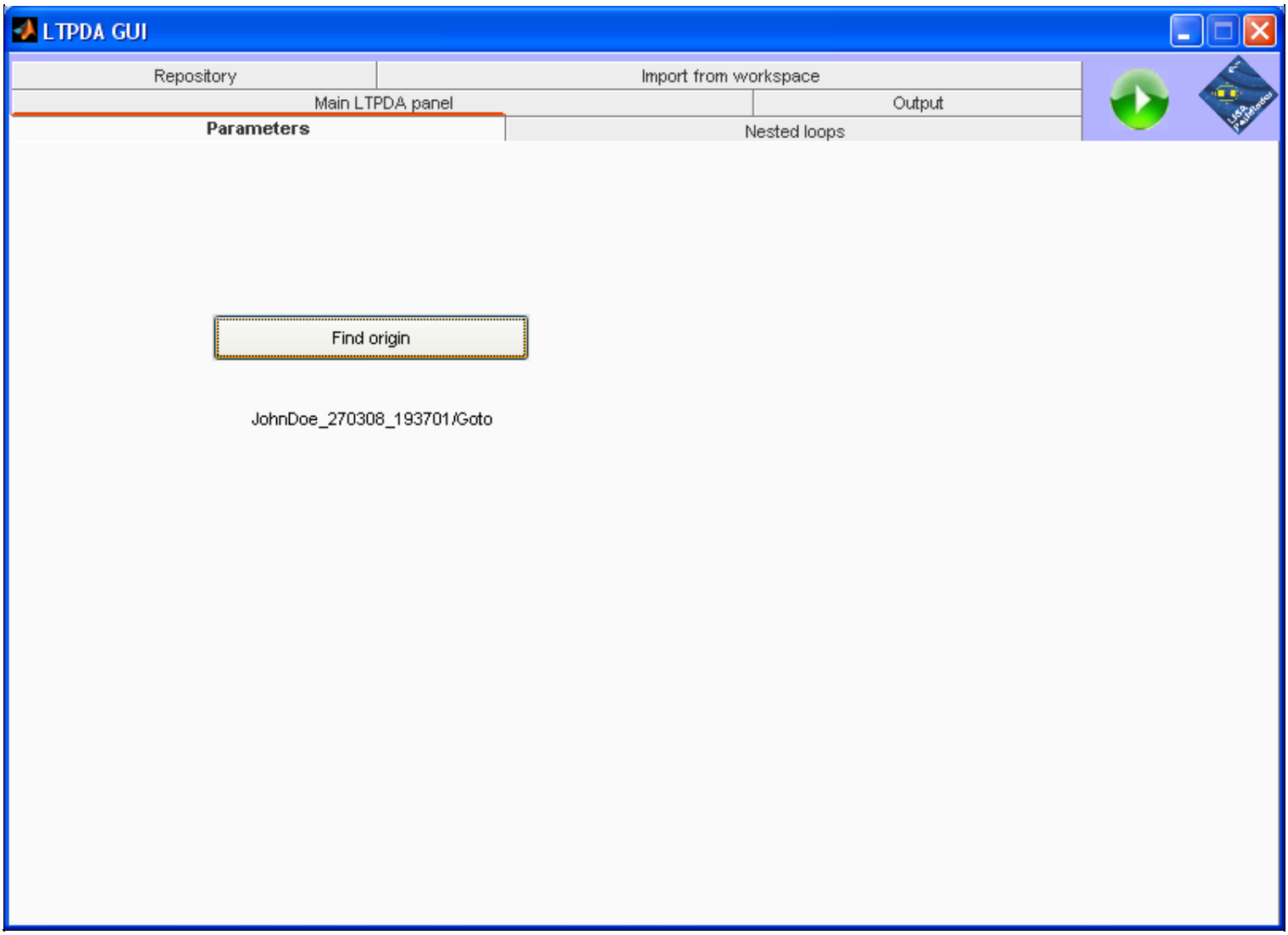


Changing this parameter will immediately alter the mux block in Simulink; unlike the direct setting of the number of inputs from Simulink, the GUI will change also the size of the block, in order to make it easier to connect the blocks and to be read.

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.

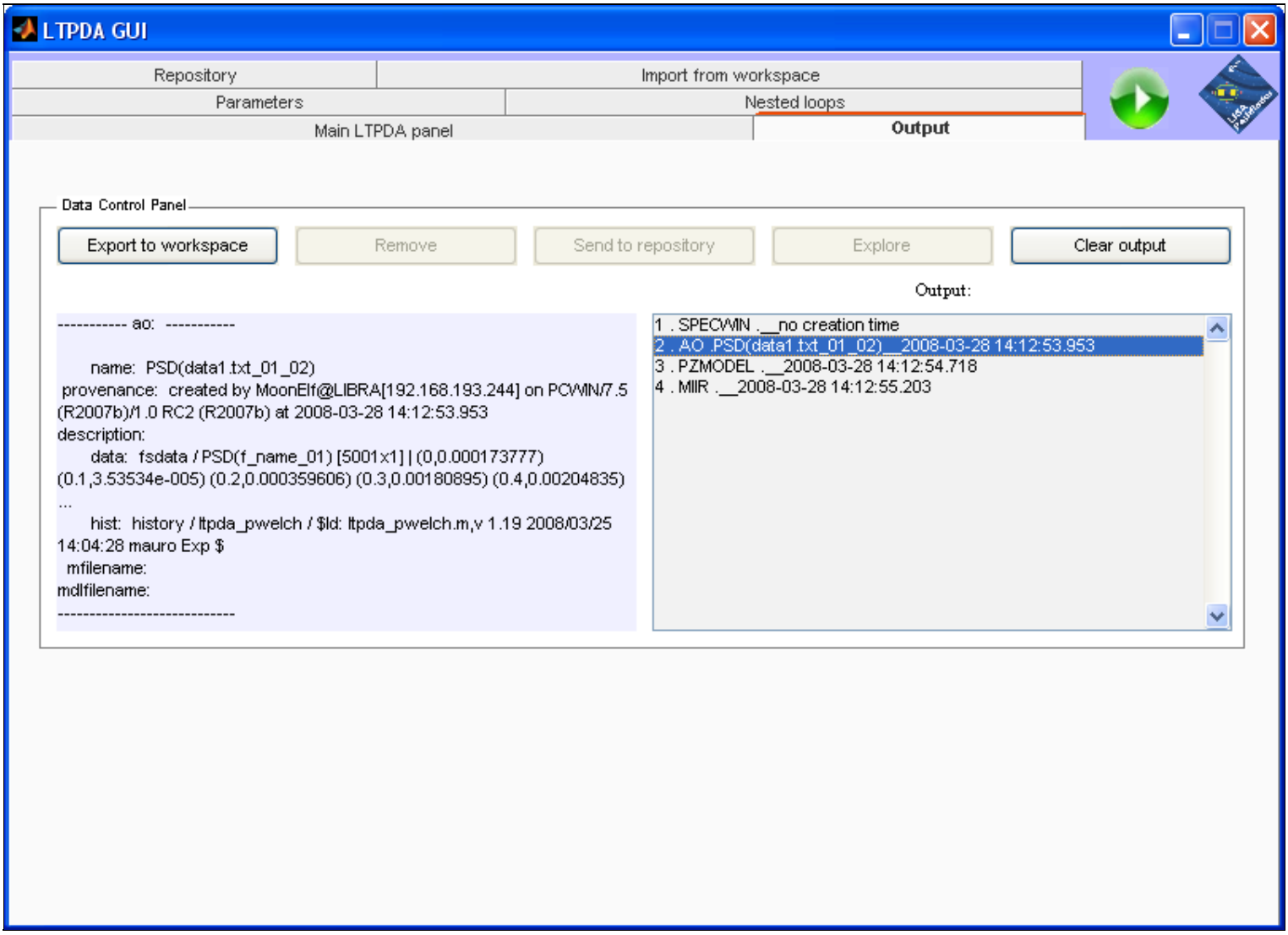
Main panel Output panel

©LTP Team

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:

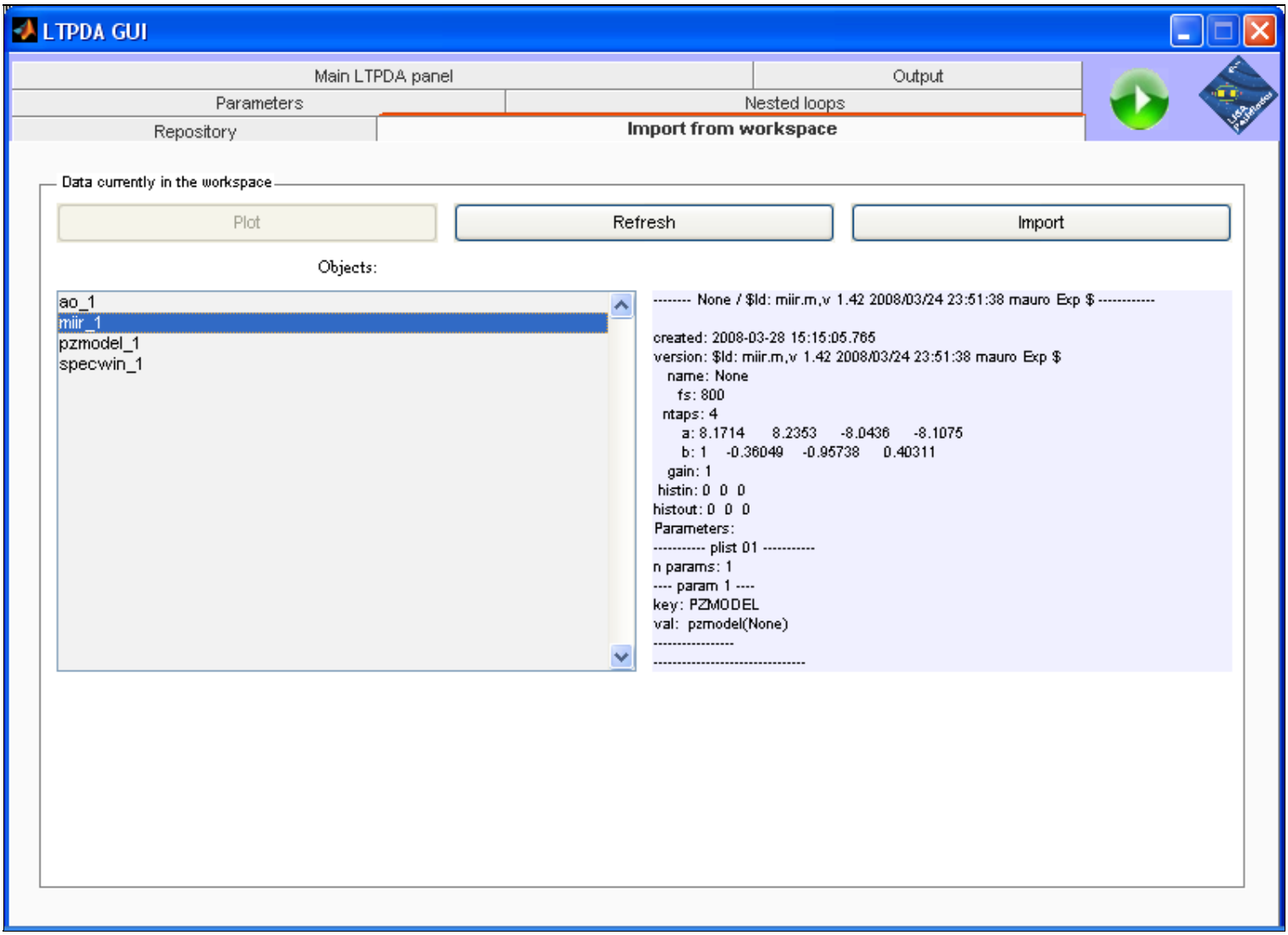
- **Export to workspace**: just like the same button in the main panel, this will save the selected object(s) into the MATLAB workspace. The name of these objects will be automatically set.
- **Remove**: *not for R1*
- **Send to repository**: *not for R1*
- **Explore**: *not for R1*
- **Clear output**: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

Output panel Repository panel

©LTP Team

Repository panel

Not meant for R1

Import panel Nested loops

©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.

Nested loops The pole/zero model helper

©LTP Team

The pole/zero model helper

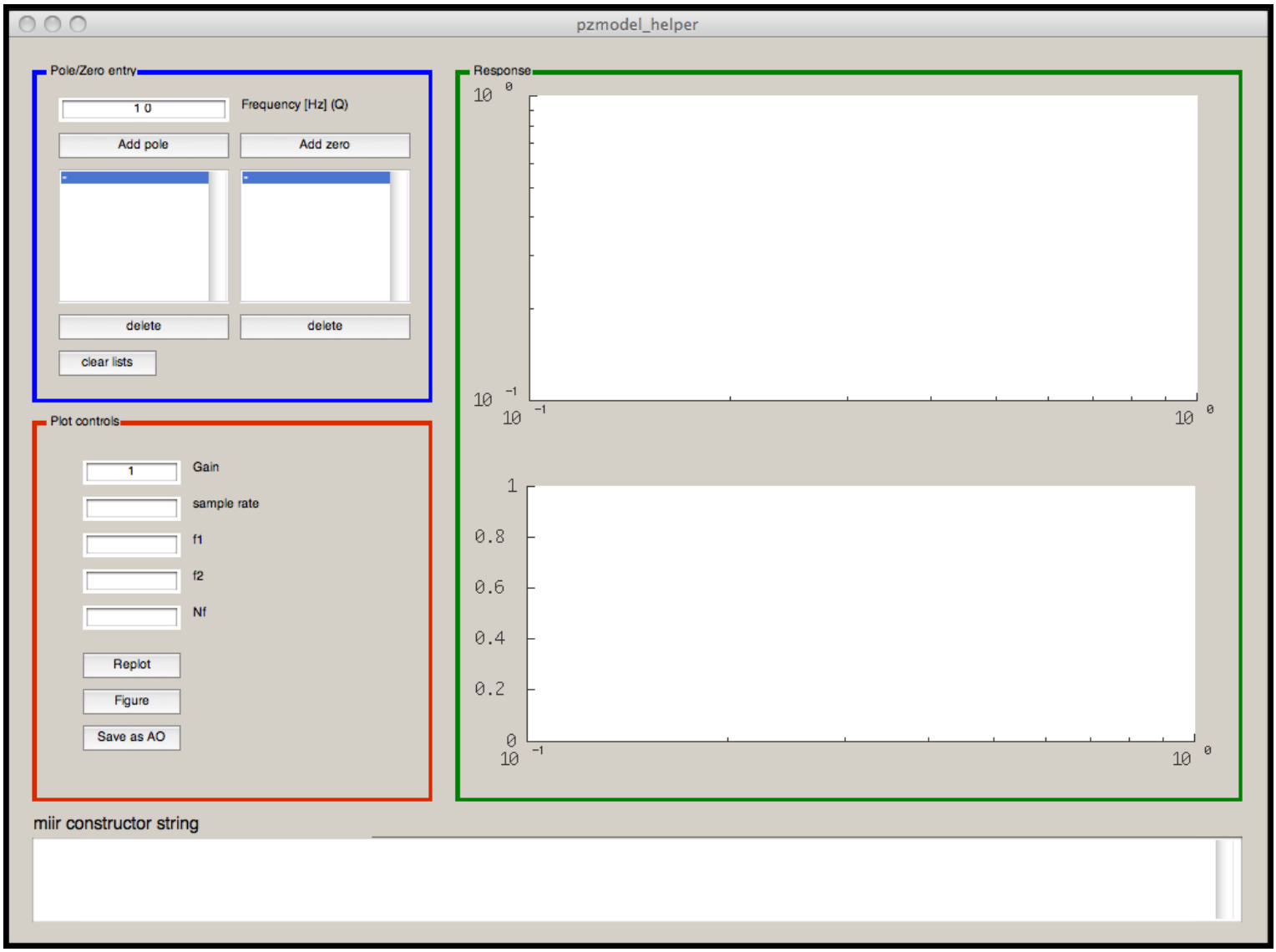
The LTPDA toolbox contains a class (`pzmodel`) for creating and using pole/zero models. The pole/zero model helper GUI allows the user to visualise the pole/zero model as it's being designed. It also allows the user to quickly see how the corresponding IIR filter (`miir` object) will look for different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

To start the pole/zero model helper:

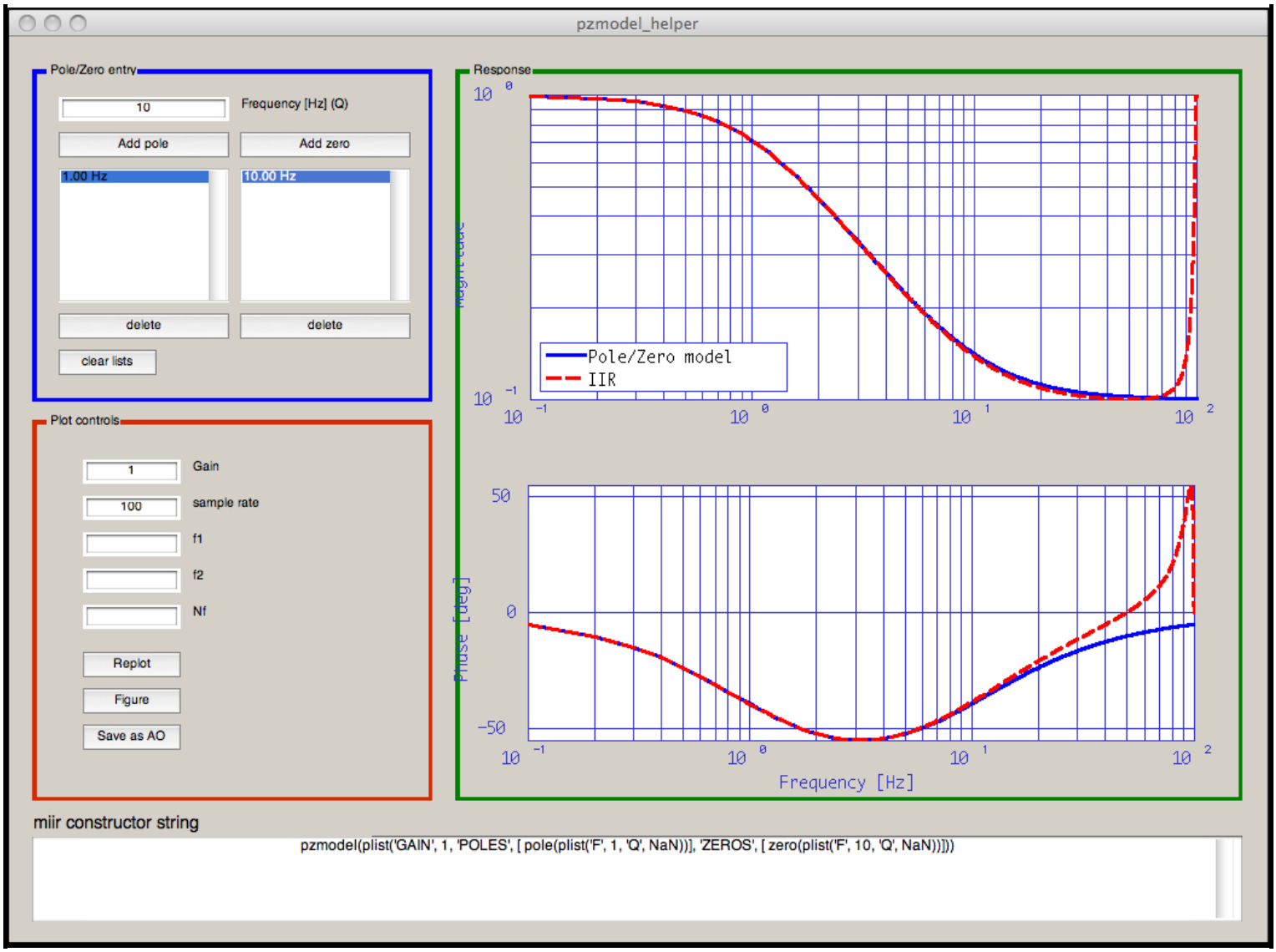
```
>> pzmodel_helper
```

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 LTPDA Repository GUI

The Spectral Window GUI

©LTP Team

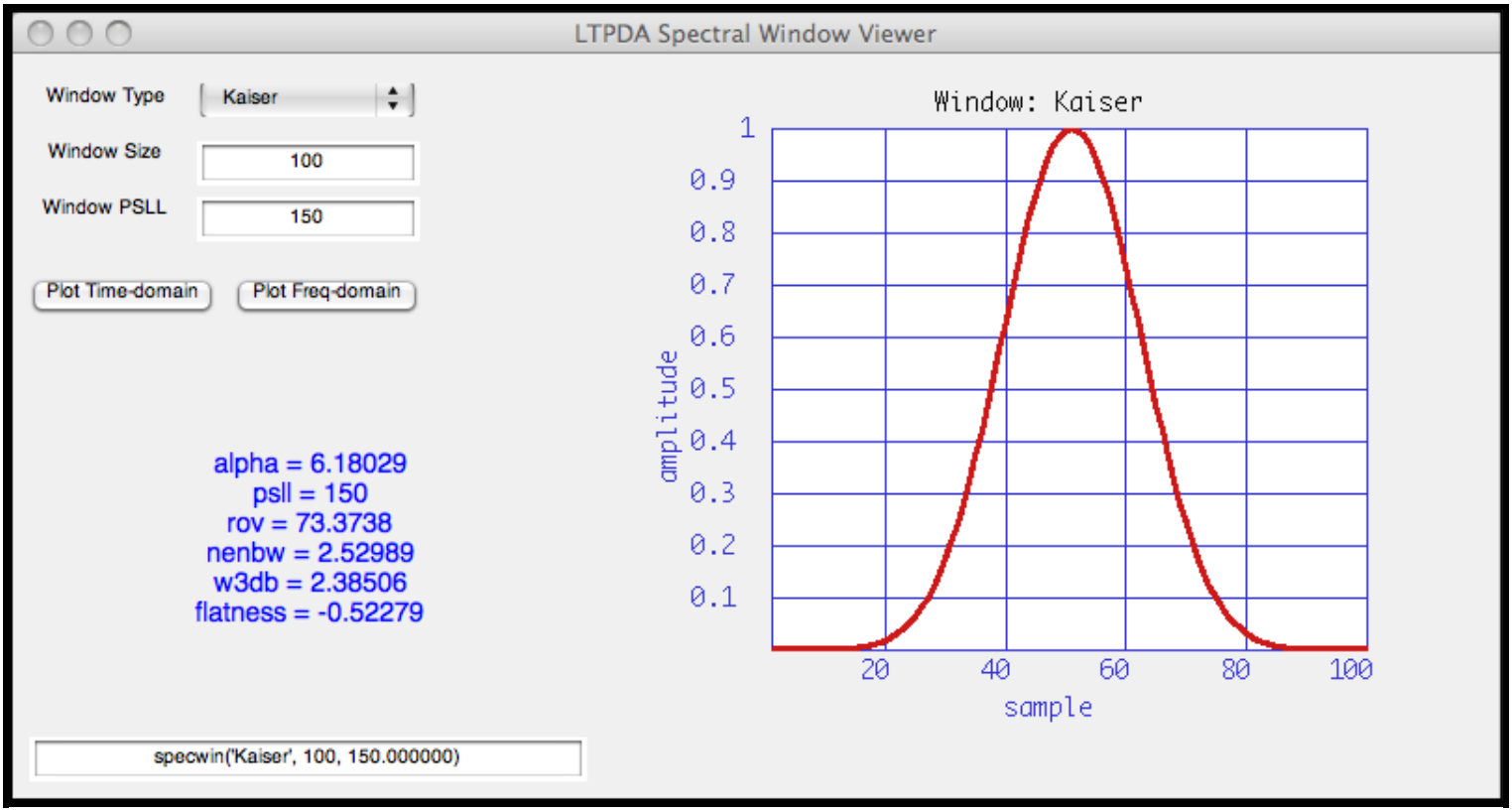
The Spectral Window GUI

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:



©LTP Team

The constructor helper

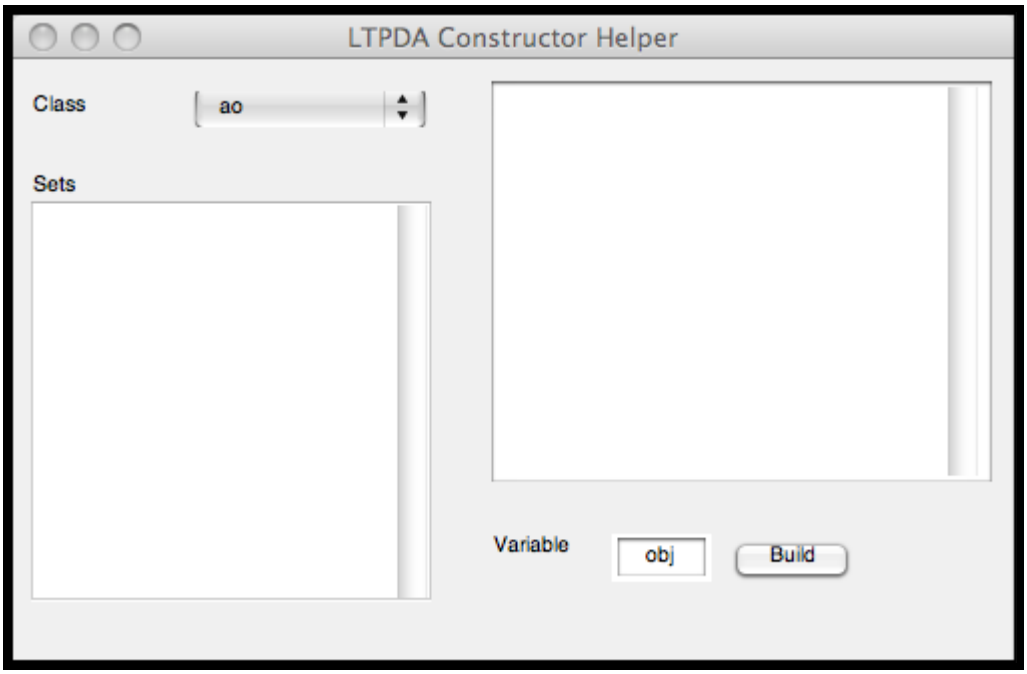
Since LTPDA is an object-oriented system, the user must create objects of different types using the appropriate constructors. The various constructor forms for each different LTPDA class can be explored using the constructor helper.

To start the constructor helper GUI:

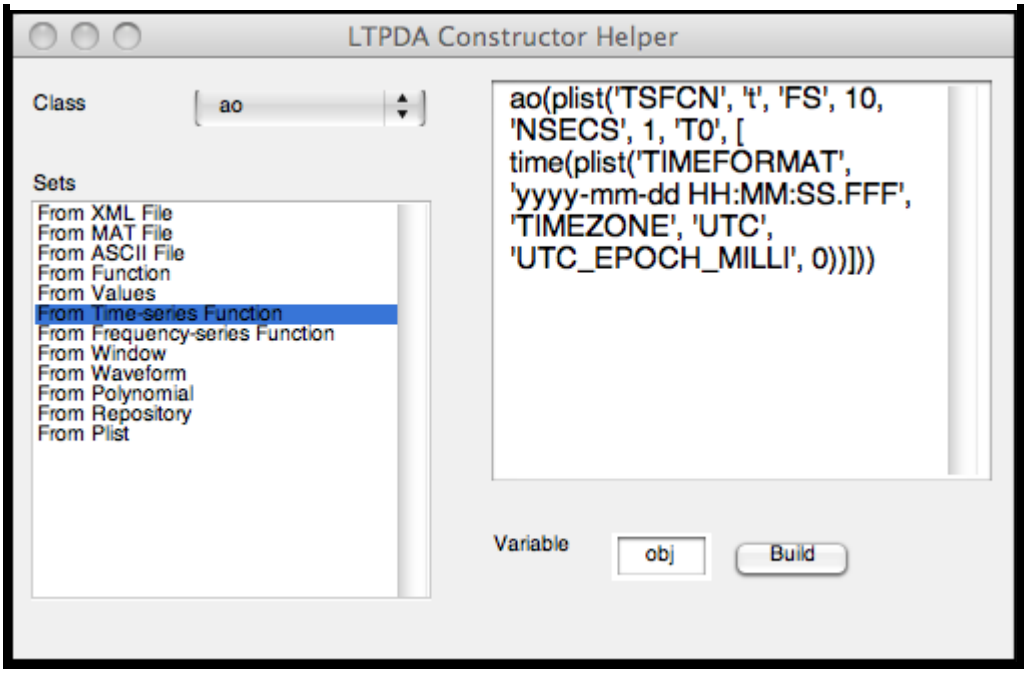
```
>> ltpda_constructor_helper
```

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

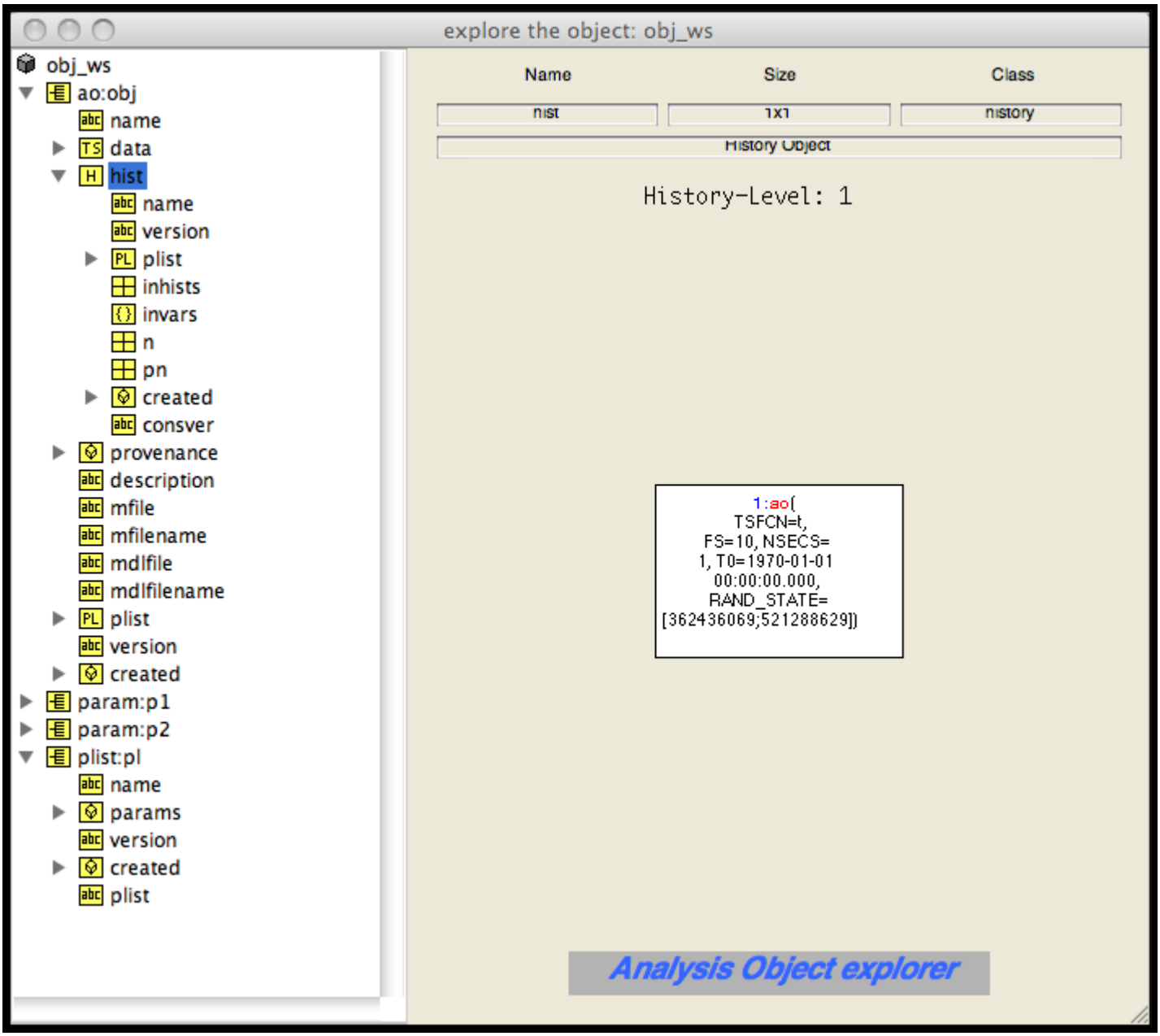
Since LTPDA works mainly with complex object types, it is often useful to explore the content of these objects graphically, particularly for Analysis Objects which may contain deep history trees. To do this, LTPDA offers the object explorer.

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

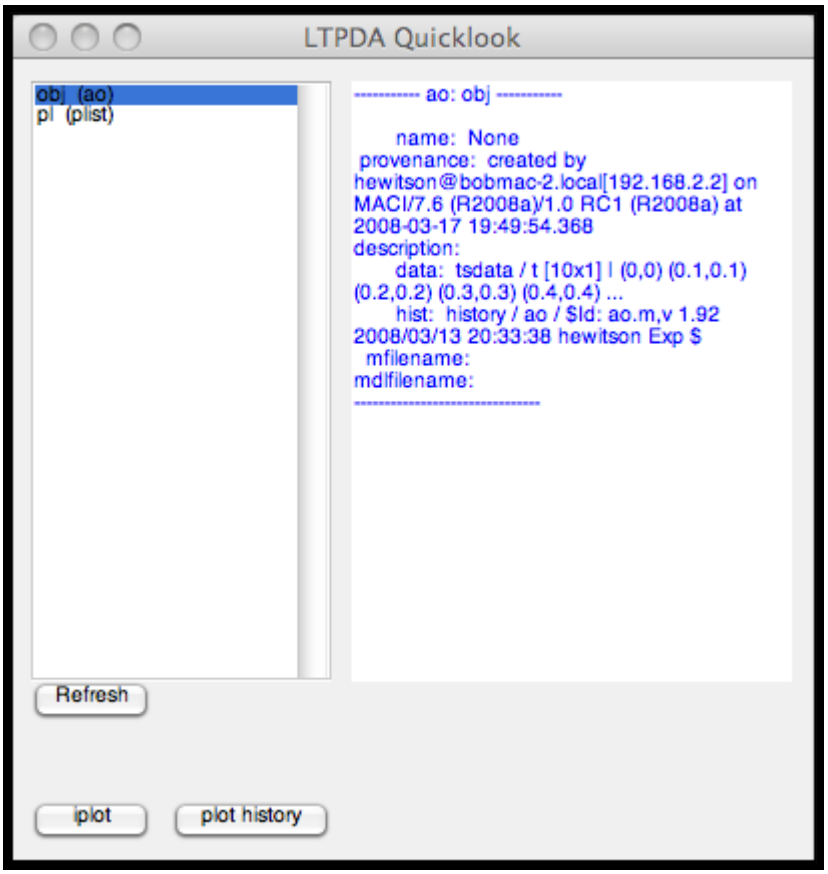
To quickly view all LTPDA objects currently in the MATLAB workspace, you can use the LTPDA Quicklook GUI.

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.

©LTP Team

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`).

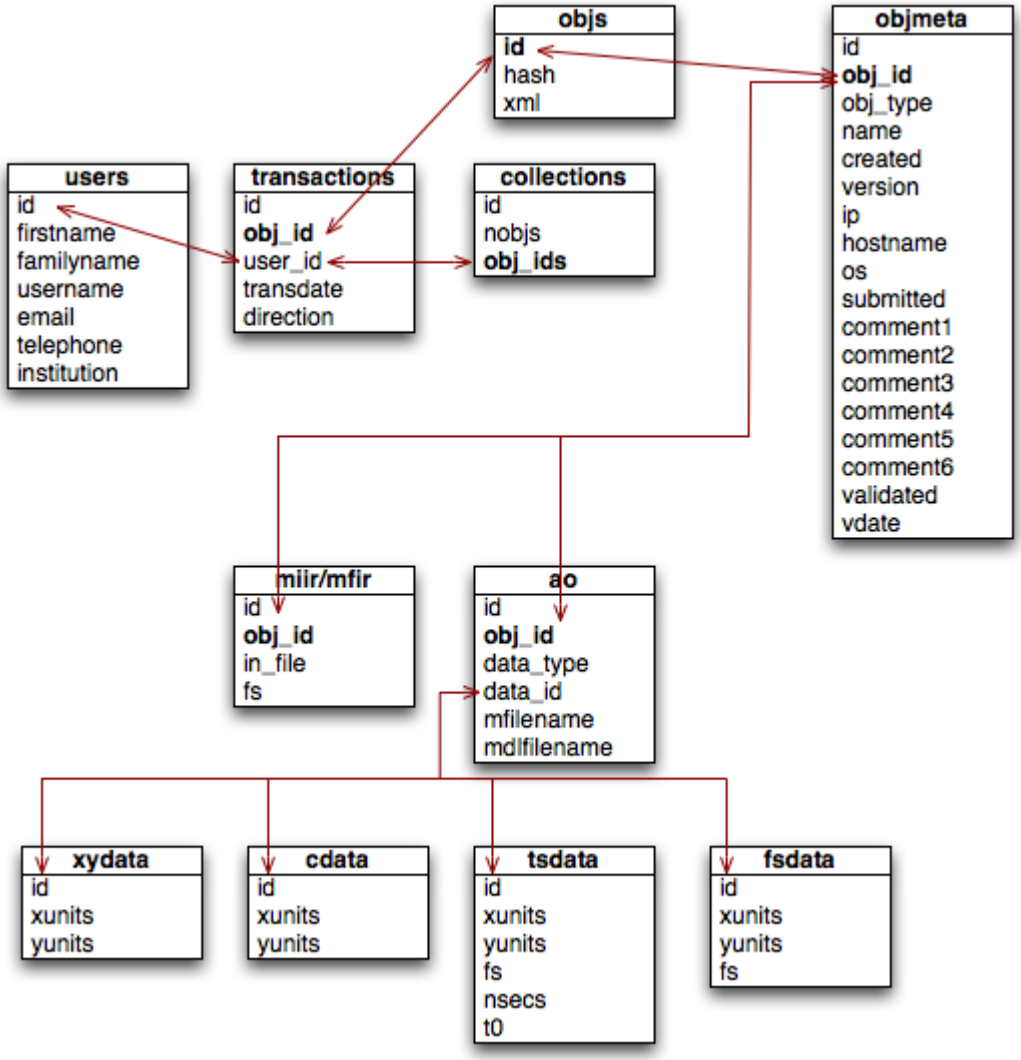
Any standard MySQL client can be used to query and search an LTPDA repository. For example, using a web-client or the standard MySQL command-line interface. In addition, the LTPDA Toolbox provides two ways to search the database: using the command `ltpda_dbquery` or using the LTPDA repository GUI. It 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 represenation of the object.
xml	longtext	The XML represenation 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 indended 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.

Working with an LTPDA Repository Connecting to an LTPDA Repository

©LTP Team

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)
```

What is an LTPDA Repository | Submitting LTPDA objects to a repository

©LTP Team

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.
```

Connecting to an LTPDA Repository Exploring an LTPDA Repository

©LTP Team

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:

ltpda_getAOsInTimeSpan	Retrieve particular AOs in the given time-span

©LTP Team

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 additon, 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
```

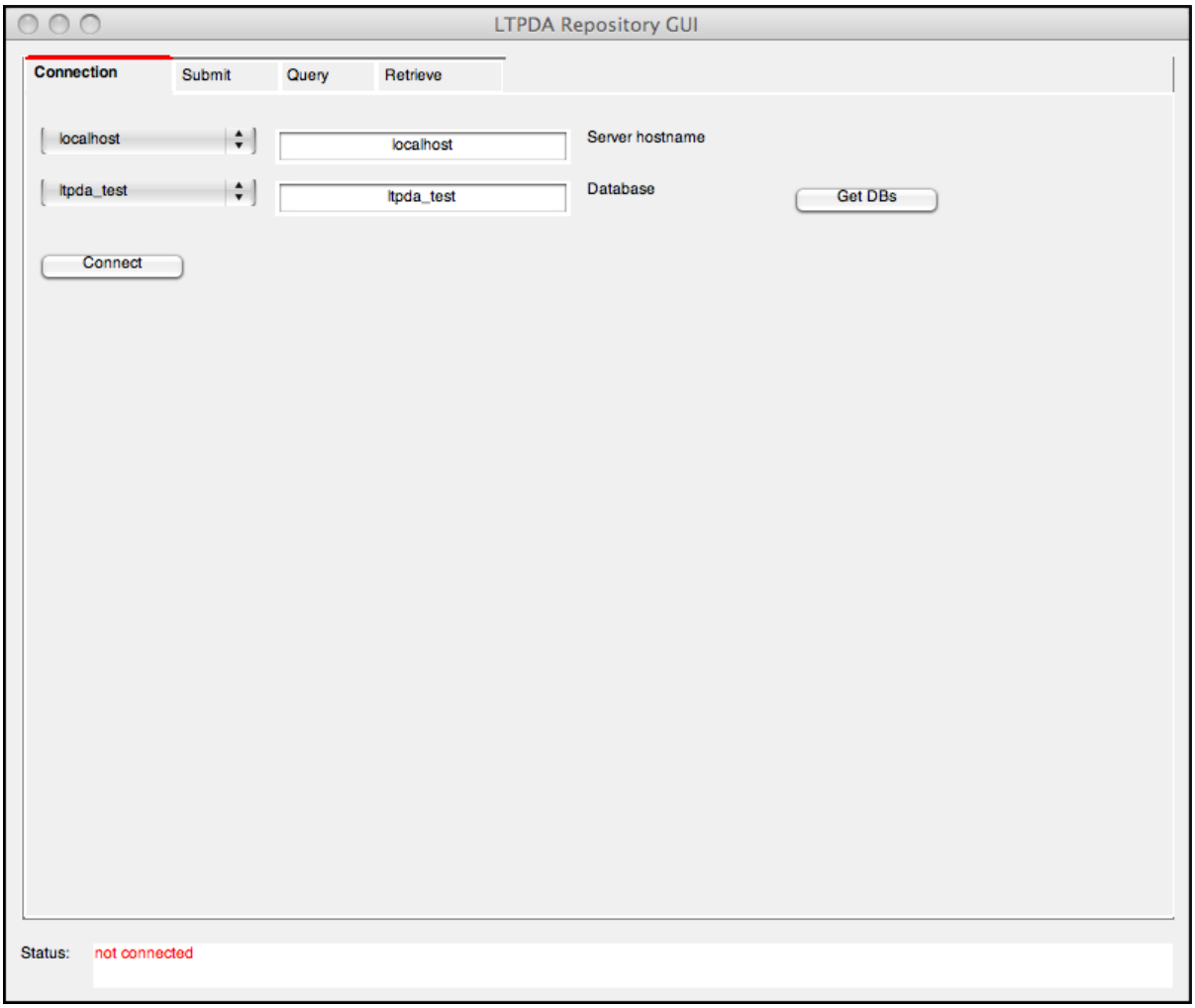
The interface allows submission and retrieval of objects, as well as querying of a repository.

Retrieving LTPDA objects from a repository Connecting to a repository

©LTP Team

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

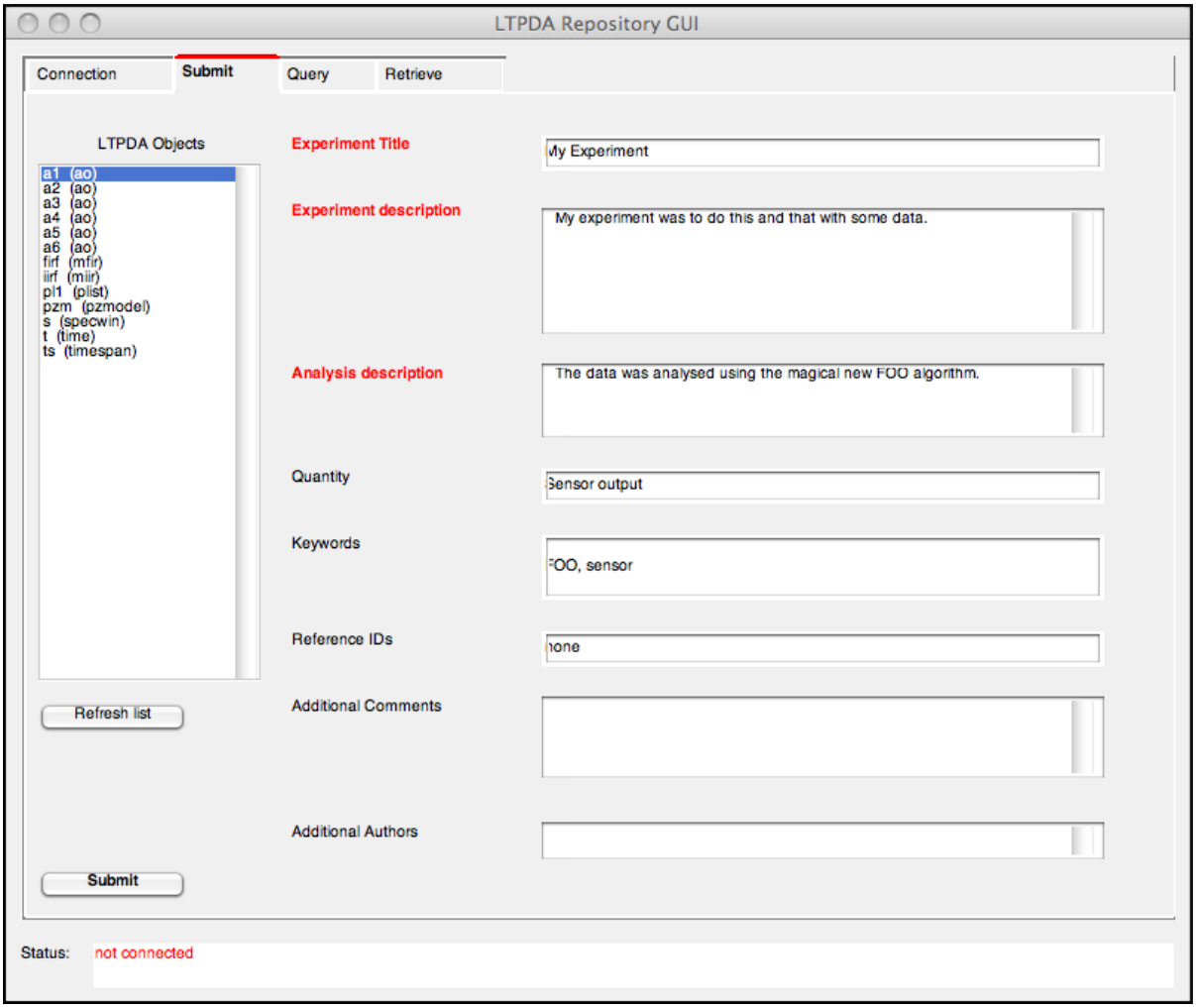
To disconnect from the current repository, click the 'Disconnect' button. This must be done before being able to connect to a different repository.

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.



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 submssion without specifying those informations will fail). The availble fields are:

- `Experiment title` A short title for the experiment from which the data originate. Mandatory field, to be filled with more that 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.

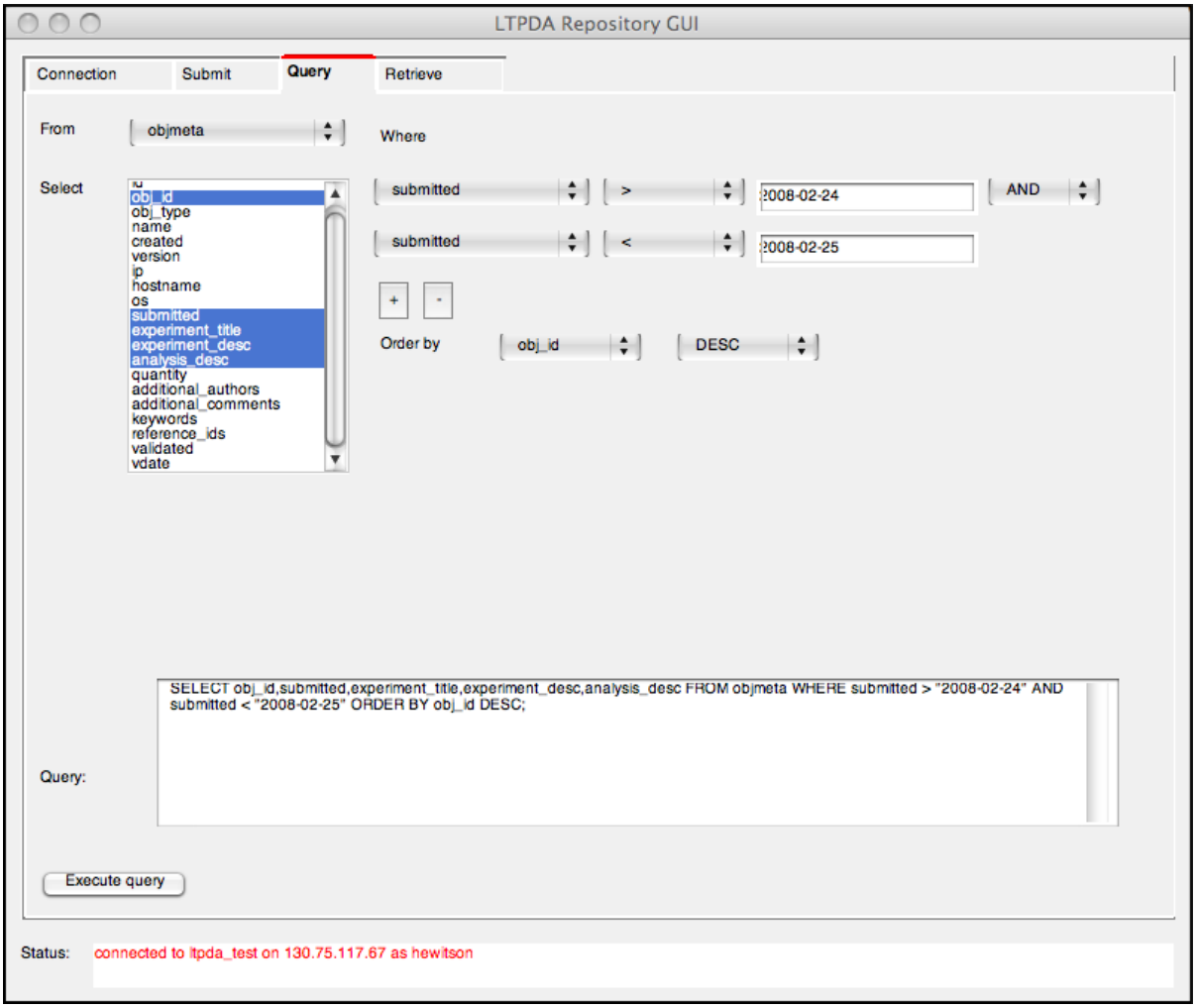
Connecting to a repository Querying the contents of a repository

©LTP Team

Querying the contents of a repository

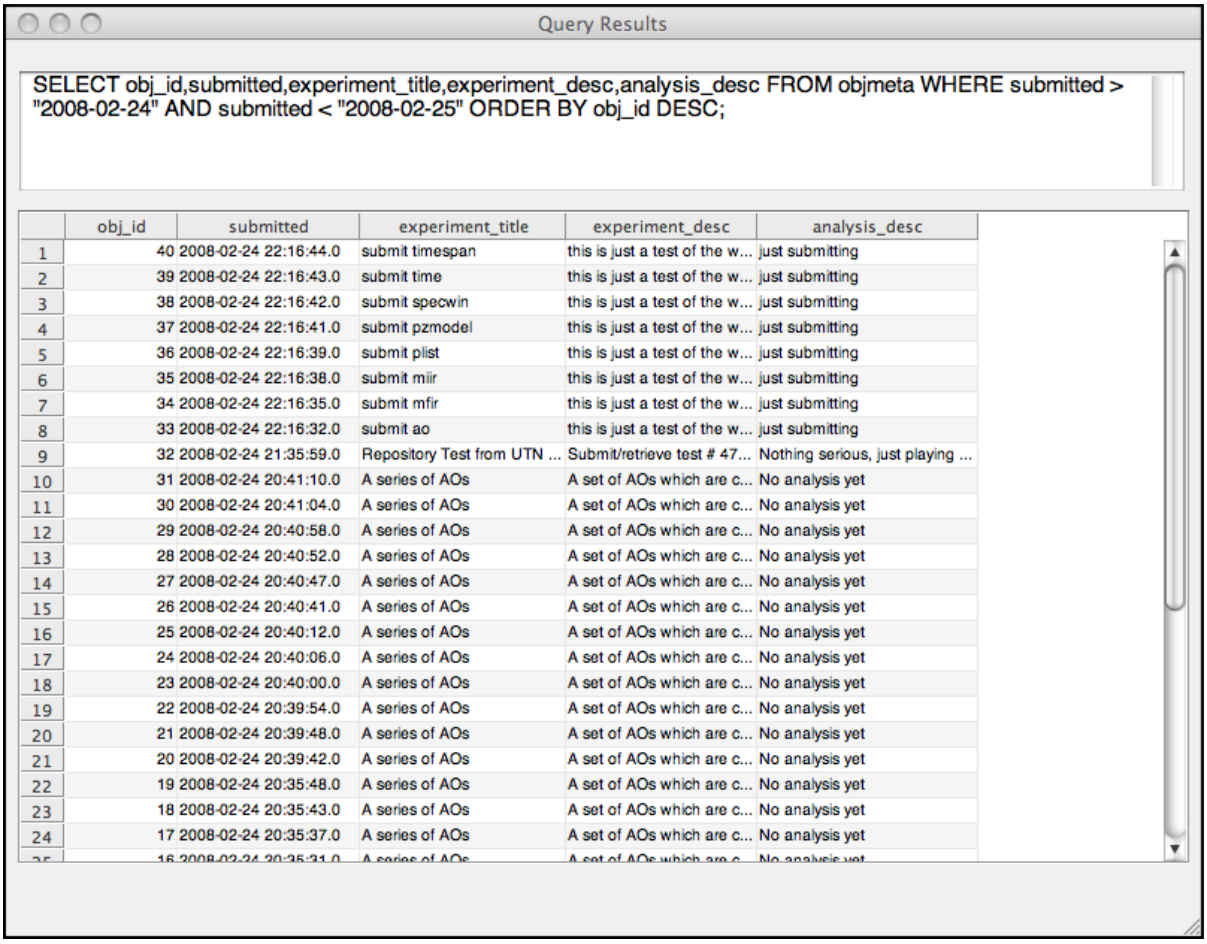
Querying an LTPDA repository is done using standard SQL statements. The repository GUI presents the user with the possibility to graphically build SQL statements which avoids learning SQL syntax. Currently, the SQL statements that can be built in this way are restricted to queries on a single database 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.



	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

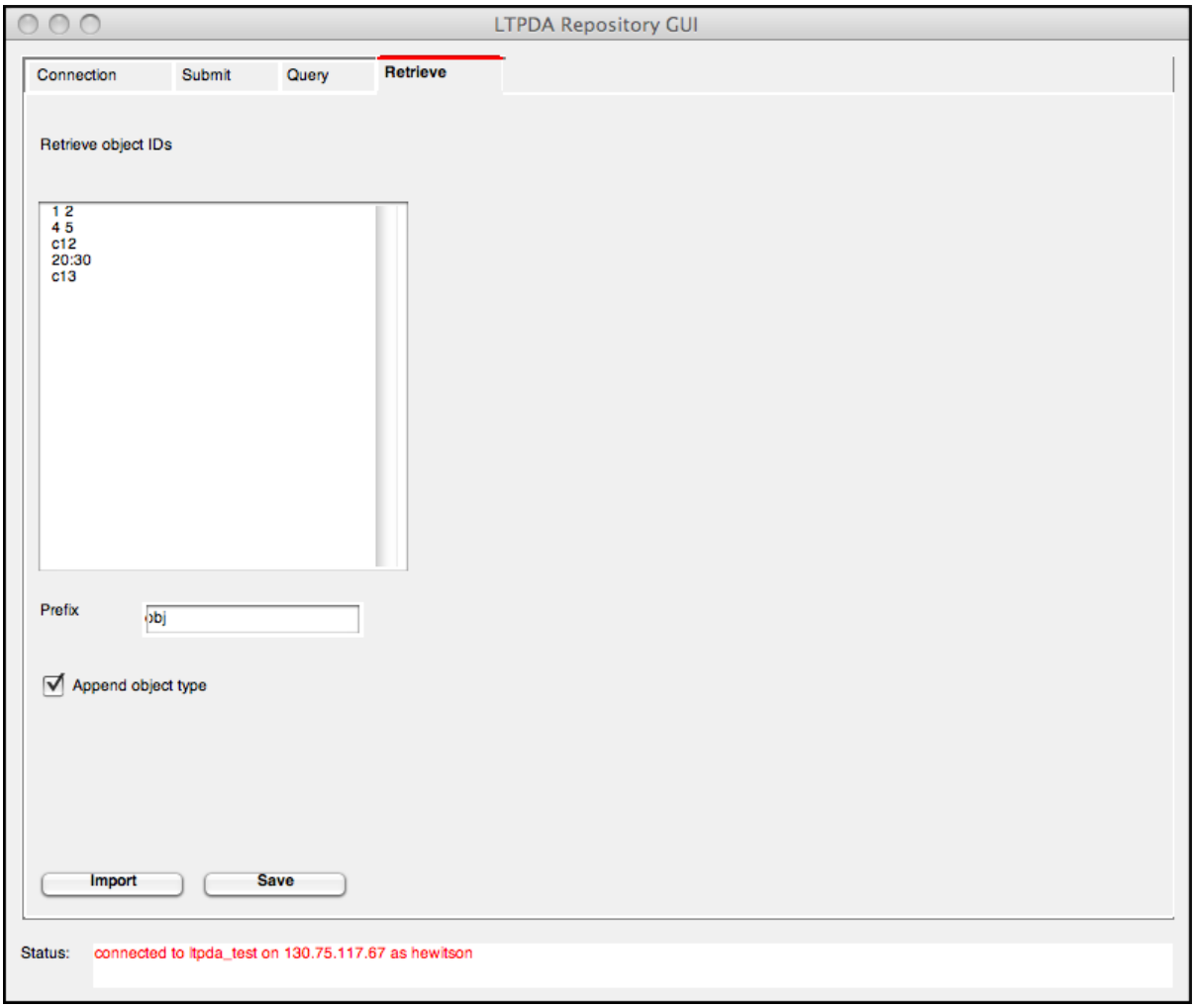
As you can see, the query string that is actually executed is presented in the text edit box above the 'Execute query' button. This query string can be edited to allow for finer control over the query. This is for users who already have a working knowledge of MySQL SQL syntax.

Submitting objects to a repository Retrieving objects and collections from a repository

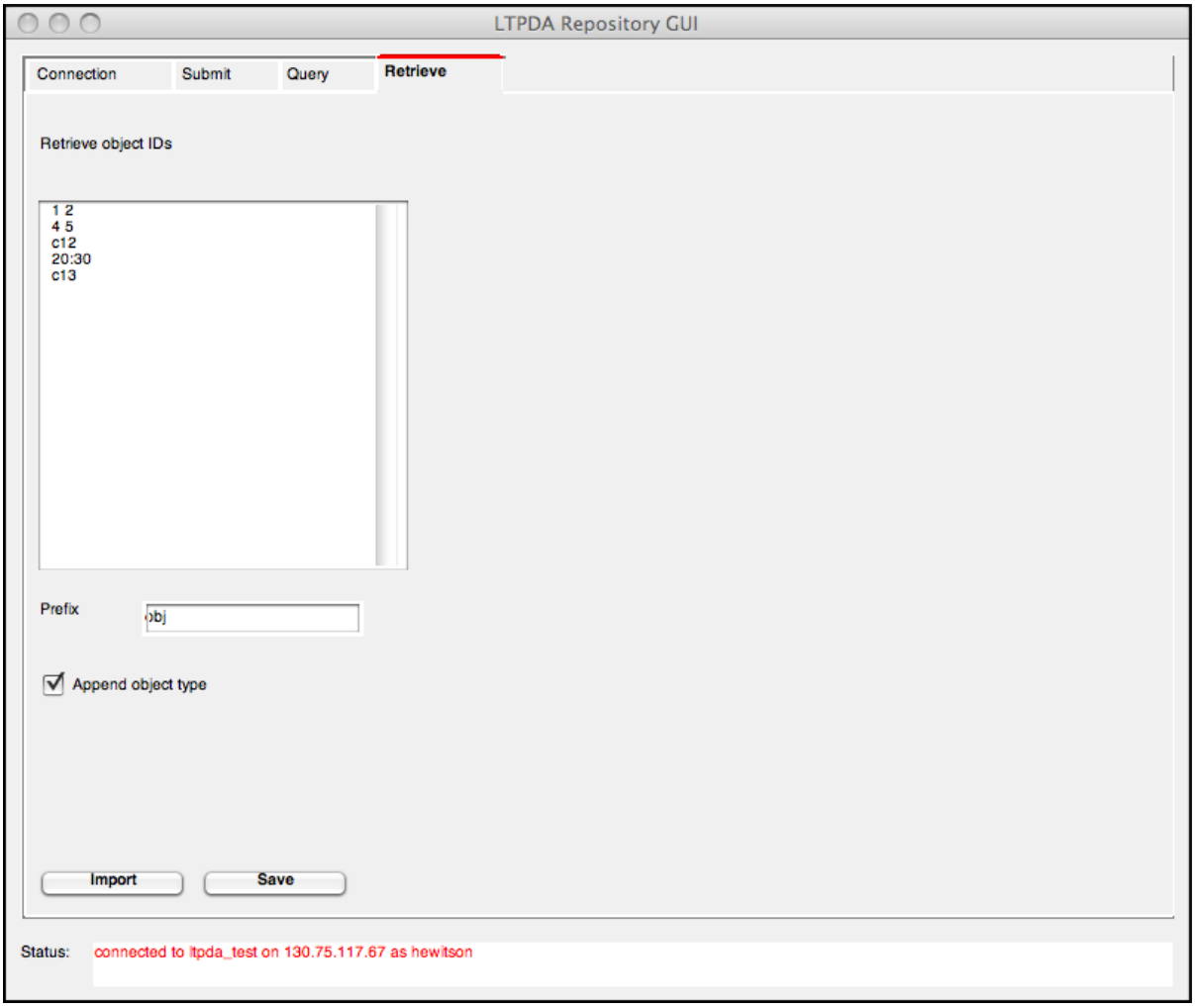
©LTP Team

Retrieving objects and collections from a repository

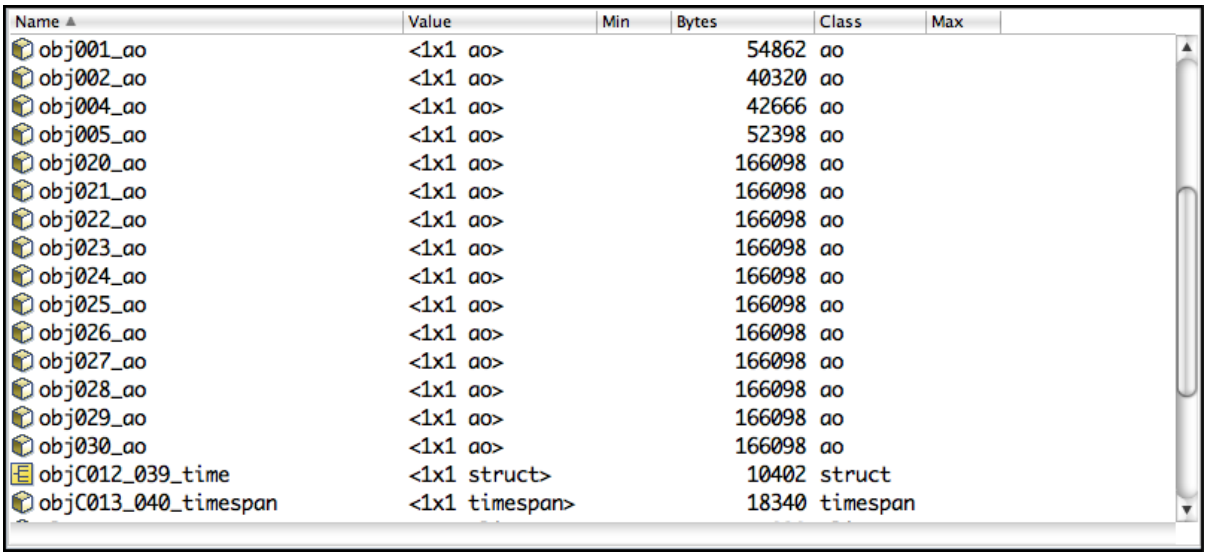
Retrieving objects from an LTPDA repository can be done using the retrieve panel shown below:



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 to acceleration with a time-domain filter	ao

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 and saves it to disk.	ao
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	DROPDUPLICATES 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 lscov 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

©LTP Team

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 subsytems, 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

ao Class mfir Class

©LTP Team

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

©LTP Team

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

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
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

mfir Class pzmodel Class

©LTP Team

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	ADDHISTORY 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 the input pzmodel.	pzmodel
resp	RESP returns the complex response of a pzmodel as an Analysis Object.	pzmodel

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

©LTP Team

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 pairor 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

©LTP Team

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

©LTP Team

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

©LTP Team

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

©LTP Team

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

©LTP Team

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

©LTP Team

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 recreate the input provenance object.	provenance

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 applys the given method to the input 2D data.	cdata
applyoperator	APPLYOPERATOR applys 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

©LTP Team

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 2D data.	data2D
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 second input	fsdata
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

©LTP Team

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 2D data.	data2D
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

©LTP Team

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 applys the given method to the input 2D data.	data2D
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 xydata objects depending of the second input	xydata

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

©LTP Team

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

xydata Class Constructor Examples

©LTP Team

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

©LTP Team

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

Constructor examples of the MFIR class

©LTP Team

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 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: 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:

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

```
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:

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

```
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)
-------------------
```

Constructor examples of the PLIST class Constructor examples of the SPECWIN class

©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

©LTP Team

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)
```

Constructor examples of the SPECWIN class Functions - By Category

©LTP Team

Functions - By Category

Categories

- Output
- Relational Operator
- Helper
- Constructor
- Internal
- Operator
- Trigonometry
- Signal Processing
- MDC01
- Arithmetic Operator
- Statespace
- STATESPACE

Output

Function name	Directory	Description
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj	% LTPDA_OBJ is the abstract ltpda base class.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj	% LTPDA_OBJ is the abstract ltpda base class.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_nuo	% LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_nuo	% LTPDA_NUO is the abstract ltpda base class for ltpda non user object classes.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo	% LTPDA_UO is the abstract ltpda base class for ltpda user object classes.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo	% LTPDA_UO is the abstract ltpda base class for ltpda user object classes.

save	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo	% SAVE overloads save operator for ltpda objects.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
report	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% REPORT generates an HTML report about the input objects.
save		
type	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% TYPE converts the input objects to MATLAB functions.
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.
factor	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit	% FACTOR factorises units in to numerator and denominator units.
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.
iplotyy	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.
report		% REPORT generates an HTML report about the input 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
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/@ltpda_data	% LTPDA_DATA is the abstract base class for ltpda data objects.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% CHAR convert a ltpda_data-object into a string.

display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% DATA2D is the abstract base class for 2-dimensional data objects.
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		
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D	% DATA3D is the abstract base class for 3-dimensional data 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/@ltpda_filter	% LTPDA_FILTER is the abstract base class for ltpda filter objects.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter	% LTPDA_FILTER is the abstract base class for ltpda filter objects.
report		% REPORT generates an HTML report about the input objects.
save		
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter	% LTPDA_FILTER is the abstract base class for ltpda filter objects.
type		
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.
report		% REPORT generates an HTML report about the input 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.
report		% REPORT generates an HTML report about the input 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.
report		% REPORT generates an HTML report about the input objects.
save		
type		
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@rational	% CHAR convert a rational object into a string.
display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@rational	% DISPLAY overloads display functionality for rational objects.
report		% REPORT generates an HTML report about the input objects.
save		
type		
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@parfrac	% CHAR convert a parfrac object into a string.

display	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@parfrac	% DISPLAY overloads display functionality for parfrac objects.
report		% REPORT generates an HTML report about the input 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.
report		% REPORT generates an HTML report about the input 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.
dotview	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% DOTVIEW view history of an object via the DOT interpreter.
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.
dotview	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% DOTVIEW view an ssm object via the DOT interpreter.
isstable	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% tells if ssm is numerically stable
report		% REPORT generates an HTML report about the input objects.
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	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj	% EQ overloads the == operator for ltpda objects.
ne	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj	% NE overloads the ~= operator for ltpda objects.
eq		
ne		
eq		
ge		
gt		
le		
lt		
ne		
eq		
ge		
gt		
le		
lt		
ne		
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		
eq		
ne		

eq		
ne		
eq		
ne		
eq		
ne		
eq		
ne		

Back to top

Helper

Function name	Directory	Description
isprop	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj	% ISPROP tests if the given field is one of the object properties.
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
setName	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo	% SETNAME Set the property 'name'.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo	% LTPDA_UO is the abstract ltpda base class for ltpda user object classes.
isprop		%ISPROP Returns true if the property exists
setName	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% SETNAME Set the property 'name'.
setProperties	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% SETPROPERTIES set different properties of an object.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% LTPDA_UOH is the abstract ltpda base class for ltpda user

		object classes with history
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'.
setPlotinfo	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SETPLOTINFO sets the 'plotinfo' property of the ao.
setProperties		% SETPROPERTIES set different properties of an object.
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).
sumjoin	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% SUMJOIN sums time-series signals togther
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 pairor 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
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
setIunits		% SETIUNITS sets the 'iunits' property of the ao.
setName		% SETNAME Set the property 'name'.
setOunits		% SETOUNITS sets the 'ounits' property of the ao.
setProperties		% SETPROPERTIES set different properties of an object.
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.

setIunits		% SETIUNITS sets the 'iunits' property of the ao.
setName		% SETNAME Set the property 'name'.
setOunits		% SETOUNITS sets the 'ounits' property of the ao.
setProperties		% SETPROPERTIES set different properties of an object.
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.
setIunits		% SETIUNITS sets the 'iunits' property of the ao.
setName		% SETNAME Set the property 'name'.
setOunits		% SETOUNITS sets the 'ounits' property of the ao.
setProperties		% SETPROPERTIES set different properties of an object.
isprop		%ISPROP Returns true if the property exists
isprop		%ISPROP Returns true if the property exists
setIunits		% SETIUNITS sets the 'iunits' property of the ao.
setName		% SETNAME Set the property 'name'.
setOunits		% SETOUNITS sets the 'ounits' property of the ao.
setProperties		% SETPROPERTIES set

		different properties of an object.
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
setIunits		% SETIUNITS sets the 'iunits' property of the ao.
setName		% SETNAME Set the property 'name'.
setOunits		% SETOUNITS sets the 'ounits' property of the ao.
setProperties		% SETPROPERTIES set different properties of an object.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@rational	% STRING writes a command string that can be used to recreate the input
isprop		%ISPROP Returns true if the property exists
setIunits		% SETIUNITS sets the 'iunits' property of the ao.
setName		% SETNAME Set the property 'name'.
setOunits		% SETOUNITS sets the 'ounits' property of the ao.
setProperties		% SETPROPERTIES set different properties of an object.
string	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@parfrac	% STRING writes a command string that can be used to recreate the input
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'.
setProperties		% SETPROPERTIES set different properties of an object.
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
findParams	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% FINDPARAMS returns parameter names matching the given pattern.
isprop		%ISPROP Returns true if the property exists
setName		% SETNAME Set the property 'name'.
setProperties		% SETPROPERTIES set different properties of an object.

Back to top

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
rebuild	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% REBUILD rebuilds the input objects using the 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	% UNIT a helper class for implementing units in LTPDA.
ao	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% AO analysis object class constructor.
rebuild		% REBUILD rebuilds the input objects using the history.
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	% XZYDATA 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.
rebuild		% REBUILD rebuilds the input objects using the history.
mfir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% MFIR FIR filter object class constructor.
rebuild		% REBUILD rebuilds the input objects using the history.
miir	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% MIIR IIR filter object class constructor.
rebuild		% REBUILD rebuilds the input objects

		using the history.
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.
rebuild		% REBUILD rebuilds the input objects using the history.
rational	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@rational	% RATIONAL rational representation of a transfer function.
rebuild		% REBUILD rebuilds the input objects using the history.
parfrac	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@parfrac	% PARFRAC partial fraction representation of a transfer function.
rebuild		% REBUILD rebuilds the input objects using the history.
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.
rebuild		% REBUILD rebuilds the input objects using the history.
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.
rebuild	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@history	% REBUILD rebuilds the orignal object using the history.
rebuild		% REBUILD rebuilds

		the input objects using the history.
ssm	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% SSM statespace model class constructor.

Back to top

Internal

Function name	Directory	Description
submit	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uo	% SUBMIT submits the given collection of objects to an LTPDA Repository.
addHistory	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh	% ADDHISTORY Add a history-object to the ltpda_uo object.
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.
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.
char	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_data	% LTPDA_DATA is the abstract base class for ltpda data objects.
applymethod	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% APPLYMETHOD applys the given method to the input 2D data.
applyoperator	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% APPLYOPERATOR applys the given operator to the two input data objects.
getX	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% GETX Get the property 'x'.
getY	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% GETY Get the property 'y'.
setX	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% SETX Set the property 'x'.
setXY	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% SETXY Set the property 'xy'.
setXunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% SETXUNITS Set the property 'xunits'.
setY	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% SETY Set the property 'y'.
setYunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data2D	% SETYUNITS Set the property 'yunits'.

applymethod	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% APPLYMETHOD applys the given method to the input 2D data.
applyoperator	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@cdata	% APPLYOPERATOR applys 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	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D	% 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	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D	% SETZ Set the property 'z'.
setZunits	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@data3D	% SETZUNITS Set the property 'zunits'.
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	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_filter	% 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.
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.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
getlowerFreq	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@rational	% GETLOWERFREQ gets the frequency of the lowest pole in the model.
getupperFreq	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@rational	% GETUPPERFREQ gets the frequency of the highest pole in the model.
submit		% SUBMIT submits the given collection of objects to an LTPDA Repository.
addHistory		% ADDHISTORY Add a history-object to the ltpda_uo object.
getlowerFreq	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@parfrac	% GETLOWERFREQ gets the frequency of the lowest pole in the model.
getupperFreq	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@parfrac	% GETUPPERFREQ gets the frequency of the highest pole in the model.
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.
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.
angle	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% ANGLE overloads the angle operator 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.
curvefit	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% CURVEFIT fit a curve to data.
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.
lscov	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LSCOV is a wrapper for MATLAB's lscov function.

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.
cov	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% COV estimate covariance of data streams.
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.
gapfillingoptim	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% GAPFILLINGOPTIM fills possible gaps in data.
heterodyne	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% HETERODYNE heterodynes time-series.
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.
lisovfit	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% LISOVFIT uses LISO to fit a pole/zero model to the input frequency-series.
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.
noisegen1D	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% NOISEGEN1D generates colored noise from withe

		noise.
noisegen2D	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% NOISEGEN2D generates cross correleted colored noise from withe noise.
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.
whiten1D	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao	% WHITEN1D whitens the input time-series.
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/@ltpda_filter	% RESP Make a frequency response of the filter.
oldresp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@mfir	% RESP Make a frequency response of the filter.
resp		% RESP shadows miir/iirResp and pzmodel/resp.
oldresp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@miir	% RESP Make a frequency response of the filter.
resp		% RESP shadows miir/iirResp and pzmodel/resp.
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.
resp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@rational	% RESP returns the complex response of a rational TF as an Analysis Object.
resp	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@parfrac	% RESP returns the complex response of a parfrac 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 subsytems, with exogenous inputs
bode	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% BODE makes a bode plot from the given inputs to outputs.
copy	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% COPY Make copy of ssm objects depending of the second input
double	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% Convert a statespace model object to double arrays for given i/o
getParamValues	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% GETPARAMVALUES

		returns parameter values for the given names.
minreal	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% minreal gives a minimal realization of a ssm object by deleting state variables
modifparams	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% modifparams enables to modifyy and substitute parameters
modiftimestep	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% modiftime modifies the timestep of a ssm object
modify	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% modify allows to exectue a string to modify a ssm object
reduce	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% reduce enables to do model simplification
reduce_model	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% REDUCE_MODEL enables to do model simplification
setparams	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% SETPARAMS enables to set parameters' value
simulate	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	%simulate simulates a discrete ssm with given inputs
ssm2dot	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% SSM2DOT converts a statespace model object a DOT file.
ssm2iirpz	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% ssm2iirpz converts a statespace model object to an miir or a pzmodel
ssm2ss	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% SSM2SS converts a statespace model object to a MATLAB statespace object.
subsparams	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	% subsparams enables to substitute symbollic patameters

Back to top

STATESPACE

Function	Directory	Description

name		
kalman	hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm	%kalman applies Kalman filtering to a discrete ssm with given i/o

Back to top

Constructor examples of the TIMESPAN class Functions - Alphabetical List

©LTP Team

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

©LTP Team

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('Hannning', 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 gauspuls)
               '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

%%%
```

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.
- lxspec 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'.
- xspec 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
- lxspec 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
- xspec 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 ^

```
% 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('Hannning', 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 gauspuls)
%                 '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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


classdef ao < ltpda_uoh


  %--------------------------------------------------
  %-------- Public (read/write) Properties  -------
  %--------------------------------------------------
  properties
  end % End (read/write) Properties

  %--------------------------------------------------
  %---------- Private read-only Properties --------
  %--------------------------------------------------
  properties (SetAccess = protected)
    data        = [];
    mfile       = '';
    mfilename   = '';
    mdlfile     = '';
    mdlfilename = '';
    procinfo    = plist();
    plotinfo    = plist();
    description = '';
    version     = '$Id: ao.m,v 1.158 2008/09/07 18:11:06 hewitson Exp $';
  end % End read only properties

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                          Check property setting                           %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = set.data(obj, val)
      if ~(isa(val, 'ltpda_data') || isempty(val))
        error('### The value for the property ''data'' must be a ltpda_data object');
      end
      obj.data = val;
    end
    function obj = set.mfile(obj, val)
      if ~ischar(val)
        error('### The value for the property ''mfile'' must be a string');
      end
      obj.mfile = val;
    end
    function obj = set.mfilename(obj, val)
      if ~ischar(val)
        error('### The value for the property ''mfilename'' must be a string');
      end
      obj.mfilename = val;
    end
    function obj = set.mdlfile(obj, val)
      if ~ischar(val)
        error('### The value for the property ''mdlfile'' must be a string');
      end
      obj.mdlfile = val;
    end
    function obj = set.mdlfilename(obj, val)
      if ~ischar(val)
        error('### The value for the property ''mdlfilename'' must be a string');
      end
      obj.mdlfilename = val;
```

```
    end
    function obj = set.procinfo(obj, val)
      if ~isa(val, 'plist')
        error('### The value for the property ''procinfo'' should be a plist.');
      end
      obj.procinfo = val;
    end
    function obj = set.plotinfo(obj, val)
      if ~isa(val, 'plist')
        error('### The value for the property ''plotinfo'' should be a plist.');
      end
      obj.plotinfo = val;
    end
    function obj = set.description(obj, val)
      if ~ischar(val)
        error('### The value for the property ''description'' should be a string.');
      end
      obj.description = val;
    end
  end


  %-------------------------------------------------
  %---------------- Private Properties -------------
  %-------------------------------------------------
  properties (GetAccess = protected, SetAccess = protected)

  end

  methods
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                               Constructor                               %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function a = ao(varargin)

      import utils.const.*
      utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

      % Check the supported version
      utils.helper.checkMatlabVersion;

      %%% Call superclass
      a = a@ltpda_uoh(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%

      %%% Collect all plists and combine them.
      [pli, invars, args] = utils.helper.collect_objects(varargin, 'plist');

      if ~isempty(pli)
        pli = pli.combine();
        %%% Append the plist to the input-arguments
        args{end+1} = pli;
      end

      %%% Execute appropriate constructor
      switch numel(args)
        case 0
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%   no input   %%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          utils.helper.msg(msg.PROC1, 'empty constructor');
          a.addHistory(ao.getInfo('ao', 'None'), [], [], []);

        case 1
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%   one input   %%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          if ischar(args{1})
            %%%%%%%%%%   a1 = ao('foo.mat')   %%%%%%%%%%
            %%%%%%%%%%   a1 = ao('foo.xml')   %%%%%%%%%%
            %%%%%%%%%%   a1 = ao('foo.txt')   %%%%%%%%%%
            %%%%%%%%%%   a1 = ao('foo.dat')   %%%%%%%%%%

            utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
            a = fromFile(a, args{1});

          elseif isa(args{1}, 'ao')
            %%%%%%%%%%   a1 = ao(ao)   %%%%%%%%%%

            utils.helper.msg(msg.PROC1, 'copying %s', args{1}.name);
            a = copy(args{1},1);
            for kk = 1:numel(args{1})
              a(kk).addHistory(ao.getInfo('ao', 'None'), [], [], a(kk).hist);
            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.procinfo    = utils.helper.struct2obj(astruct.procinfo, '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, 'pzmodel');
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
```

```
                %-------------------------------------------------------
                %--- Construct from Time-series function
                %-------------------------------------------------------
                utils.helper.msg(msg.PROC2, 'constructing from fcn(t) %s', tsfcn);
                a = fromTSfcn(a, pl);

              elseif ~isempty(fsfcn)

                %-------------------------------------------------------
                %--- Construct from frequency-series function
                %-------------------------------------------------------
                utils.helper.msg(msg.PROC2, 'constructing from fcn(f) %s', fsfcn);
                a = a.fromFSfcn(pl);

              elseif ~isempty(win)

                %-------------------------------------------------------
                %--- Construct from Window
                %-------------------------------------------------------
                utils.helper.msg(msg.PROC2, 'constructing from window %s', char(win));
                a = a.fromSpecWin(pl);

              elseif ~isempty(waveform)

                %-------------------------------------------------------
                %--- Construct from Waveform
                %-------------------------------------------------------
                utils.helper.msg(msg.PROC2, 'constructing from waveform %s', waveform);
                a = fromWaveform(a, pl);

              elseif ~isempty(hostname) || ~isempty(conn)

                %-------------------------------------------------------
                %--- Construct from repository
                %-------------------------------------------------------
                utils.helper.msg(msg.PROC2, 'constructing from repository %s', hostname);
                a = a.fromRepository(pl);

              elseif ~isempty(polycoeffs)

                %-------------------------------------------------------
                %--- Construct from polynomial
                %-------------------------------------------------------
                utils.helper.msg(msg.PROC2, ['constructing from polynomial '
mat2str(polycoeffs)]);
                a = a.fromPolyval(pl);

              elseif ~isempty(ipl)

                %-------------------------------------------------------
                %--- Construct from plist
                %-------------------------------------------------------
                % if the plist is empty, we return an empty ao
                if nparams(ipl) == 0
                else
                  % do plist constructor
                  a = ao(ipl);
                end

              elseif ~isempty(pzm)

                %-------------------------------------------------------
                %--- Construct from pzmodel
                %-------------------------------------------------------
                utils.helper.msg(msg.PROC2, 'constructing from pzmodel %s', char(pzm));
                a = a.fromPzmodel(pl);

              else
                %-------------------------------------------------------
                %--- ERROR
                %-------------------------------------------------------
                if nparams(pl) == 0
                  %%% is the plist is empty then return an empty AO
                  a = ao();
                else
                  error('### Unknown AO constructor method.');
                end
              end

            elseif isa(args{1}, 'specwin')
              %%%%%%%%%%   a1 = ao(specwin)   %%%%%%%%%%

              utils.helper.msg(msg.PROC1, 'constructing from spectral window %s',
char(args{1}));
              a = a.fromSpecWin(args{1});

```

```
        elseif isa(args{1}, 'ltpda_data')
          %%%%%%%%%%   a1 = ao(ltpda_data-object)   %%%%%%%%%%
          %%%%%%%%%%   a1 = ao(cdata)               %%%%%%%%%%
          %%%%%%%%%%   a1 = ao(fsdata)              %%%%%%%%%%
          %%%%%%%%%%   a1 = ao(tsdata)              %%%%%%%%%%
          %%%%%%%%%%   a1 = ao(xydata)              %%%%%%%%%%
          %%%%%%%%%%   a1 = ao(xyzdata)             %%%%%%%%%%

          utils.helper.msg(msg.PROC1, 'constructing from data object %s',
class(args{1}));
          a            = ao;
          a.data       = args{1};
          a.addHistory(ao.getInfo('ao', 'None'), [], [], []);

        else
          error('### Unknown single input constructor');
        end

      case 2
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%   two inputs   %%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if isa(varargin{1}, 'database')
          %%%%%%%%%%   ao(database-object, [IDs])   %%%%%%%%%%

          utils.helper.msg(msg.PROC1, 'constructing from database object');
          pl = plist('conn', varargin{1}, 'id', varargin{2});
          a = a.fromRepository(pl);

        elseif isnumeric(args{1}) && isnumeric(args{2})
          %%%%%%%%%%   ao(x-vector, y-vector)   %%%%%%%%%%

          utils.helper.msg(msg.PROC1, 'constructing from X and Y values');
          a = a.fromVals(plist('XVALS', args{1}, 'YVALS', args{2}));

        elseif ischar(args{1}) && isa(args{2}, 'plist')
          %%%%%%%%%%%   ao('foo.txt', pl)   %%%%%%%%%%

          utils.helper.msg(msg.PROC1, 'constructing from filename and plist');
          pl = combine(plist('filename', args{1}), args{2});
          a = a.fromFile(pl);

        elseif isa(args{1}, 'ao') && isa(args{2}, 'ao')
          %%%%%%%%%%%   ao(ao-object, ao-object)   %%%%%%%%%%
          % Do we have a list of AOs as input
          a = ao([args{1}, args{2}]);

        elseif isa(args{1}, 'ao') && isa(args{2}, 'plist') && isempty(args{2}.params)
          % pass to copy constructor
          a = ao(args{1});
        elseif isa(args{1}, 'ao') && isa(args{2}, 'plist')
          %%%%%%%%%%%   ao(ao-object, plist-object)   %%%%%%%%%%
          % This constructor is necessary for the function type.
          % Ignore the first AO and build a new AO from the plist.

          a = ao(args{2});
        else
          error('### Unknown constructor with two inputs');
        end

      otherwise
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%   other inputs   %%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        [aoi, invars, rest] = utils.helper.collect_objects(args, 'ao');

        %%% Do we have a list of AOs as input
        if ~isempty(aoi) && isempty(rest)
          a = ao(aoi);
        else
          error('### Unknown number of arguments.');
        end
    end
  end % End constructor
end % End public methods


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              Methods (static)                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
methods (Static=true)
  varargout = mdc1_input_noises(varargin)
  varargout = update_struct(varargin);

```

```
    function out = SETS()
      out = {...
        'Default', ...
        'From XML File', 'From MAT File', 'From ASCII File',...
        'From Function', 'From Values',...
        'From Time-series Function', 'From Frequency-series Function', ...
        'From Window', 'From Waveform', 'From Polynomial', 'From Repository',...
        'From Plist', 'From Pzmodel'};
    end

    function out = VEROUT()
      out = '$Id: ao.m,v 1.158 2008/09/07 18:11:06 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'ao');
    end

    % Return the plist for a particular parameter set
    function out = getDefaultPlist(set)

      % Otherwise we try to find a set for this constructor
      switch set
        case 'Default'
          out = plist();

          %-----------------------------------------
          %--- Repository constructor
          %-----------------------------------------
        case 'From Repository'
          out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);

          %-----------------------------------------
          %--- Read from XML file
          %-----------------------------------------
        case 'From XML File'
          out = plist('filename', '');

          %-----------------------------------------
          %--- Read from MAT file
          %-----------------------------------------
        case 'From MAT File'
          out = plist('filename', '');

          %-----------------------------------------
          %--- Read from ASCII file
          %-----------------------------------------
        case 'From ASCII File'
          out = plist(...
            'filename',    '',...
            'type',        'tsdata',...
            'columns',      [1 2],...
            'xunits',        unit('s'), ...
            'yunits',        unit('V'), ...
            't0',            time(0),...
            'comment_char', '', ...
            'use_fs',        '', ...
            'Robust', 'yes');

          %-----------------------------------------
          %--- Create from a function description
          %-----------------------------------------
        case 'From Function'
          out = plist('fcn', 'randn(100,1)');

          %-----------------------------------------
          %--- Create from a set of values
          %-----------------------------------------
        case 'From Values'
          out = plist('vals', 1, 'N', 1);

          %-----------------------------------------
          %--- Create from a time-series function
          %-----------------------------------------
        case 'From Time-series Function'
          out = plist('tsfcn', 't', 'fs', 10, 'nsecs', 1, 't0', time(0));

          %-----------------------------------------
          %--- Create from frequency-series function
          %-----------------------------------------
        case 'From Frequency-series Function'
          out = plist('fsfcn', 'f', 'f1', 1e-9, 'f2', 5, 'nf', 1000, 'scale', 'log',
'f', []);

          %-----------------------------------------
          %--- Create from a window function
```

```
            %-----------------------------------------
          case 'From Window'
            out = plist('win', specwin('Hanning', 100));

            %-----------------------------------------
            %--- Create from a set of polynomial coefficients
            %-----------------------------------------
          case 'From Polynomial'
            out = plist('polyval', [-1e-4 .02 -1 -1], 'Nsecs', 10, 'fs', 10, 't', []);

            %-----------------------------------------
            %--- Create from a plist
            %-----------------------------------------
          case 'From Plist'
            out = plist('Plist', plist);

            %-----------------------------------------
            %--- Create from a waveform description
            %-----------------------------------------
          case 'From Waveform'
            out = plist('waveform', 'sine wave',...
              'A', 1, ...
              'f', 1.23,...
              'phi', 0,...
              'fs', 10,...
              'nsecs', 10,...
              't0', time(0));

            %-----------------------------------------
            %--- Create from a set of polynomial coefficients
            %-----------------------------------------
          case 'From Pzmodel'
            out = plist('pzmodel', pzmodel, 'Nsecs', 0, 'fs', 0);
          otherwise
            out = plist();
        end
      end % End getDefaultPlist
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                          Methods (public)                                 %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    methods
      % Getters
      val = xunits(varargin)
      val = yunits(varargin)
      val = fs(varargin)
      val = t0(varargin)
      val = x(varargin)
      val = y(varargin)
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                          Methods (static, private)                        %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    methods (Static=true, Access=private)
      % constructor functions
      % Spectral estimate function
      [yy, info] = welchscale(xx, win, fs, norm, inunits)
      varargout = welch(varargin)
      [x,M,isreal_x,y,Ly,win,winName,winParam,noverlap,k,L,options] = welchparse(x,esttype,varargin)
      [P,f] = computeperiodogram(x,win,nfft,esttype,varargin)
      [Xx,f] = computeDFT(xin,nfft,varargin)
      varargout = xspec(varargin)

      % Noise generator functions
      varargout = ngconv(varargin)
      varargout = ngsetup(varargin)
      varargout = ngsetup_vpa(varargin)
      varargout = nginit(varargin)
      varargout = ngprop(varargin)
      varargout = fq2fac(varargin)
      varargout = conv_noisegen(varargin)
      varargout = mchol(varargin)

      % LPSD-type methods
      varargout = ltf_plan(varargin)
      varargout = mlpsd_mex(varargin)
      varargout = mlpsd_m(varargin)
      varargout = mltfe(varargin)
      varargout = findFsMax(varargin)
      varargout = findShortestVector(varargin)
      varargout = lxspec(varargin)
    end
```

```


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                               Methods (public)                               %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods

    % Setters
    varargout = setXunits(varargin)
    varargout = setYunits(varargin)
    varargout = setT0(varargin)
    varargout = setFs(varargin)
    varargout = setXY(varargin)
    varargout = setY(varargin)
    varargout = setX(varargin)
    varargout = setDescription(varargin)
    varargout = setZ(varargin)


    % Other methods
    varargout = copy(varargin)
    h = md5(varargin)
  end % End public methods


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (protected)                             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Access = protected)
    obj = fromDataInMAT(obj, data, filename)
    obj = fromDatafile(obj, pli)
  end


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                             Methods (private)                                %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Access=private)
    % Constructors
    a = fromVals(a, pli)
    a = fromTSfcn(a, pli)
    a = fromWaveform(a, pli)
    a = fromFcn(a,pli)
    a = fromFSfcn(a,pli)
    a = fromSpecWin(a,pli)
    a = fromPolyval(a,pli)
    a = fromPzmodel(a,pli)
    % Others
    varargout = applymethod(varargin)
    varargout = applyoperator(varargin)
  end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 pairor 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

%%%
```

- pset PSET set or add a key/value pairor 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 ^

```
% 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 pairor 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef plist < ltpda_uo

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                              %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    params   = [];
  end

  %---------- Protected Properties ----------
  properties (SetAccess = protected)
    version = '$Id: plist.m,v 1.51 2008/09/07 18:15:26 hewitson Exp $';
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                             Check property setting                            %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                  Constructor                                  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function pl = plist(varargin)

      import utils.const.*
      utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);

      %%% Call superclass
      pl = pl@ltpda_uo(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      %%%%%%%%%%%   pl = plist()   %%%%%%%%%%
      % create default parameter list object
      if nargin == 0
        utils.helper.msg(msg.OPROC1, 'empty constructor');
      elseif nargin == 1

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
      %%%%%%%%%%   p = param(plist)   %%%%%%%%%%
      if isa(varargin{1}, 'plist')
        pli = varargin{1};

        %%% if the plist is empty then return an empty plist object
        if isempty(varargin{1}.params)
        else

          % Retrieve from repository?
          hostname = find(pli, 'hostname');
          if ~isempty(hostname)
            utils.helper.msg(msg.OPROC1, 'retrieving from repository %s', hostname);
            pl = pl.fromRepository(pli);
          else
            utils.helper.msg(msg.OPROC1, 'copying from %s', varargin{1}.name);
            pl = copy(varargin{1}, 1);
          end
        end

        %%%%%%%%%%%% From file %%%%%%%%%%%%%%%%%%%%%%%%
      elseif ischar(varargin{1})

        filename = varargin{1};
        utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
        pl = fromFile(pl, filename);

%         [path, name, ext] = fileparts(filename);
%         switch ext
%           case '.mat'
%             pl = load(filename);
%             pl = pl.a;
%           case '.xml'
%             root_node = xmlread(filename);
%             pl = utils.helper.xmlread(root_node, 'plist');
%           otherwise
%             error('### Unknown file type.');
%         end

        %%%%%%%%%%   pl = plist(struct)   %%%%%%%%%%
      elseif isstruct(varargin{1})

        utils.helper.msg(msg.OPROC1, 'constructing from struct');
        %%% Set properties which are declared in this class
        pl.params = utils.helper.struct2obj(varargin{1}.params, 'param');

        %%%%%%%%%%   pl = plist(param)   %%%%%%%%%%
      elseif isa(varargin{1}, 'param')

        utils.helper.msg(msg.OPROC1, 'constructing from param');
        %%% Use only upper case characters for the 'key'
        param_list = varargin{1};
        for kk = 1:length(param_list)
          param_list(kk) = param_list(kk).setKey(upper(param_list(kk).key));
        end

        pl.params = param_list;

      else
        error ('### unknown arguments to construct a parameter list')
      end

    elseif nargin == 2

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   two inputs   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      %%%%%%%%%%%   From DATABASE   %%%%%%%%%%%
      if isa(varargin{1}, 'database')

        utils.helper.msg(msg.OPROC1, 'constructing from database object');
        pl = retrieve(varargin{1}, varargin{2:end});

        %%%%%%%%%%   pl = plist('key1', val1)   %%%%%%%%%%
      else
        utils.helper.msg(msg.OPROC1, 'constructing from key/value pair');
        pl.params = param(upper(varargin{1}), varargin{2});
      end

    elseif nargin > 2

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   more inputs   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
        utils.helper.msg(msg.OPROC1, 'constructing from key/value pairs');

        %%%%%%%%%%   pl = plist('key1, val1, 'key2', 'val2' , ...)   %%%%%%%%%%
        param_list = [];
        argin  = varargin;
        while length(argin) >= 2
          key = upper(argin{1});
          val =       argin{2};
          argin = argin(3:end);

          if ~isempty(pl.params)
            found = any(strcmpi(key, {pl.params(:).key}));
            if found
              error('### Do not use the same key [%s] twice.\n### REMARK: The key is not case sensitive.', key);
            end
          end
          % add to list
          param_list = [param_list param(key, val)];
        end
        pl.params  = param_list;
      else
        error('### Unknown number of constructor arguments.');
      end

    end % End constructor
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                               Methods (public)                             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    %%% Define Abstract methods
    varargout = char(varargin);
    varargout = copy(varargin)
    varargout = display(varargin);
    varargout = string(varargin);

    %%% Define other methods
    varargout = append(varargin);
    varargout = combine(varargin);
    varargout = find(varargin);
    varargout = isparam(varargin);
    varargout = nparams(varargin);
    varargout = pset(varargin);
    %     varargout = pzmresp(varargin);
    varargout = remove(varargin)
    varargout = resp(varargin);
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                               Methods (protected)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Access = protected)
    varargout = fromRepository(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                           Methods (Static, Public)                         %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true)

    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: plist.m,v 1.51 2008/09/07 18:15:26 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'plist');
    end

    function out = SETS()
      out = {'Default', 'From Repository', 'From MAT File'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        case 'From Repository'
          out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
        case 'From MAT File'
```

```
          out = plist('filename', '');
        otherwise
          error('### Unknown set [%s] to get the default plist.', set);
      end
    end

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                          Methods (Static, Private)                        %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true, Access = private)
  end

end % Class


```

Generated on Mon 08-Sep-2008 13:18:47 by ***m2html*** *© 2003*

param

PURPOSE ^

PARAM Parameter object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 PARAM Parameter object class constructor.
%%%
```

```
 SEE ALSO: ltpda_obj, ltpda_nuo, plist

%%%
```

```
%       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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef (Sealed = true) param < ltpda_nuo

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                            %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    key     = '';
    val     = [];
  end

  %---------- Protected Properties ----------
  properties (SetAccess = protected)
    version = '$Id: param.m,v 1.44 2008/09/03 16:35:22 hewitson Exp $'
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                            Check property setting                           %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = set.key(obj, val)
      if ischar(val)
        obj.key = upper(val);
      else
        error('### The value for the property ''key'' must be a string\n### but it is from the class %s', class(val));
      end
    end
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                 Constructor                                 %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = param(varargin)

      %%% Call superclass
      obj = obj@ltpda_nuo(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if nargin == 0

        % Do nothing

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%   one input   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      elseif nargin == 1

        %%%%%%%%%%  obj = param(param)   %%%%%%%%%%
        if isa(varargin{1}, 'param')
          obj = copy(varargin{1}, 1);

          %%%%%%%%%%  obj = param(plist)   %%%%%%%%%%
        elseif isa(varargin{1}, 'plist')

          %%%%%%%%%%  obj = param(plist())   %%%%%%%%%%
          %%%%%%%%%%  obj = param(plist('KEY', 'a', 'VAL', 1))   %%%%%%%%%%
          %%% is the plist is empty then return an empty param object
          if nparams(varargin{1}) == 0

          else
            pl = varargin{1};
            key = find(pl, 'key');
            val = find(pl, 'val');
            if isempty(key)
              error('### building a parameter from a plist requires one parameter in the plist is called ''key''');
            end
            if isempty(val)
              error('### building a parameter from a plist requires one parameter in the plist is called ''val''');
            end

            obj.key  = key;
            obj.val  = val;
          end

          %%%%%%%%%%  obj = param(struct)   %%%%%%%%%%
        elseif isstruct(varargin{1})
          %%% Set properties which are declared in this class
          obj.key     = varargin{1}.key;
          obj.val     = varargin{1}.val;
          obj.version = varargin{1}.version;

          %%%%%%%%%%  obj = param('filename.xml')   %%%%%%%%%%
        elseif ischar(varargin{1})

          filename = varargin{1};
          [path, name, ext] = fileparts(filename);
          switch ext
            case '.mat'
              obj = load(filename);
              obj = obj.a;
            case '.xml'
              root_node = xmlread(filename);
              obj = utils.helper.xmlread(root_node, 'param');
            otherwise
              error('### Unknown file type.');
          end

        else
          error('### unknown constructor type for param object.');
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%   two inputs   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      elseif nargin == 2

          obj.key = varargin{1};
          obj.val = varargin{2};

      else
        error('### Unknown number of arguments.');
      end

    end
  end


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                               Methods (public)                            %
```

```
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    varargout = mux(varargin);
    varargout = string(varargin)

    %%% Define Abstract methods
    varargout = char(varargin)
    varargout = copy(varargin)
    varargout = display(varargin)

    varargout = setKey(varargin)
    varargout = setVal(varargin)
    varargout = setKeyVal(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                               Methods (protected)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Access = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                           Methods (Static, Public)                         %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true)

    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: param.m,v 1.44 2008/09/03 16:35:22 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'param');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          error('### Unknown set [%s]', set');
      end
    end

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                           Methods (Static, Private)                        %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true, Access = private)
  end

end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %%%%%%%%%%%%%%%%%%%%

- 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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef pzmodel < ltpda_uoh

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                           Property definition                              %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
```

```
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    gain    = NaN;
    poles   = pz;
    zeros   = pz;
    version = '$Id: pzmodel.m,v 1.60 2008/09/07 10:48:46 hewitson Exp $';
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %---------- Abstract Properties ----------
  properties (Abstract = true, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Check property setting                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = set.gain(obj, val)
      if ~isnumeric(val) || isempty(val) || ~isreal(val)
        error('### The value for the property ''gain'' must be a real number');
      end
      obj.gain = val;
    end
    function obj = set.poles(obj, val)
      if ~isa(val, 'pz') && ~isempty(val)
        error('### The value for the property ''poles'' must be an array of pz objects.');
      end
      obj.poles = val;
    end
    function obj = set.zeros(obj, val)
      if ~isa(val, 'pz') && ~isempty(val)
        error('### The value for the property ''zeros'' must be an array of pz objects.');
      end
      obj.zeros = val;
    end
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                    Constructor                               %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = pzmodel(varargin)

      import utils.const.*
      utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);

      % Call superclass
      obj = obj@ltpda_uoh(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%

      % Collect all pzmodel objects
      [pzs, invars, rest] = utils.helper.collect_objects(varargin(:), 'pzmodel');

      if isempty(rest) && ~isempty(pzs)
        % Do copy constructor and return
        utils.helper.msg(msg.OPROC1, 'copy constructor');
        obj = copy(pzs, 1);
        for kk=1:numel(obj)
          obj(kk).addHistory(pzmodel.getInfo('pzmodel', 'None'), [], [], obj(kk).hist);
        end
```

```
        return
      end

      switch nargin
        case 0
          utils.helper.msg(msg.OPROC1, 'empty constructor');
          obj.addHistory(pzmodel.getInfo('pzmodel', 'None'), [], [], []);
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%% Zero inputs %%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        case 1
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%% One inputs %%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          if ischar(varargin{1})
            %======== Filename
            %%%%%%%%%%   pzm = pzmodel('foo.mat')   %%%%%%%%%%
            %%%%%%%%%%   pzm = pzmodel('foo.xml')   %%%%%%%%%%
            %%%%%%%%%%   pzm = pzmodel('foo.liso')  %%%%%%%%%%
            utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
            obj = fromFile(obj, varargin{1});

          elseif isstruct(varargin{1})
            %======== Struct
            %%%%%%%%%%   pzm = pzmodel(struct)   %%%%%%%%%%

            %%% Set properties which are declared in this class
            utils.helper.msg(msg.OPROC1, 'constructing from struct');
            pzm_struct = varargin{1};

            obj.gain    = pzm_struct.gain;
            obj.poles   = utils.helper.struct2obj(pzm_struct.poles, 'pz');
            obj.zeros   = utils.helper.struct2obj(pzm_struct.zeros, 'pz');
            obj.version = varargin{1}.version;

          elseif isa(varargin{1}, 'plist')
            %======== Plist
            %%%%%%%%%%   pzm = pzmodel(plist)   %%%%%%%%%%
            pl       = varargin{1};

            % Selection of construction method
            if pl.isparam('filename')
              utils.helper.msg(msg.OPROC1, 'constructing from file %s', varargin{1});
              % construct from file
              obj = fromFile(obj, pl);
            elseif pl.isparam('hostname') || pl.isparam('conn')
              utils.helper.msg(msg.OPROC1, 'constructing from repository %s', pl.find('hostname'));
              % do hostname constructor
              obj = obj.fromRepository(pl);
            elseif pl.isparam('gain') || pl.isparam('poles') || pl.isparam('zeros')
              utils.helper.msg(msg.OPROC1, 'constructing from poles and zeros');
              % Construct from poles and zeros
              obj = fromPolesAndZeros(obj, pl);
            elseif pl.isparam('ipl')
              ipl = find(pl, 'plist');
              % if the plist is empty, we return an empty PZMODEL
              if ipl.nparams == 0
              else
                % do plist constructor
                obj = pzmodel(ipl);
              end
            else
              %%% if the plist is empty then return empty PZMODEL
              if nparams(pl) == 0
              else
                error('### Unknown PZMODEL constructor method.');
              end
```

```
            end
          else
            error('### Unknown single argument constructor.');
          end

        case 2
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%% Two inputs %%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          if isa(varargin{1}, 'database') && isnumeric(varargin{2})

            %%%%%%%%%%  f = pzmodel(<database-object>, [IDs])   %%%%%%%%%%
            % pzmodel(<database-object>, [IDs])
            utils.helper.msg(msg.OPROC1, 'retrieve from repository');
            obj = obj.fromRepository(plist('conn', varargin{1}, 'id', varargin{2}));
          elseif isa(varargin{1}, 'pzmodel') && isa(varargin{2}, 'plist') && isempty(varargin{2}.params)
            % pass to copy constructor
            obj = pzmodel(varargin{1});
          else
            error('### Unknown 2 argument constructor.');
          end

        case 3
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%% Three inputs %%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          %%%%%%%%%%   pzm = pzmodel(gain, poles, zeros)   %%%%%%%%%%
          utils.helper.msg(msg.OPROC1, 'constructing from poles and zeros');
          pl = plist('gain', varargin{1}, 'poles', varargin{2}, 'zeros', varargin{3});
          obj = fromPolesAndZeros(obj, pl);
        case 4
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%% Four inputs %%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          %%%%%%%%%%   pzm = pzmodel(gain, poles, zeros, name)   %%%%%%%%%%
          utils.helper.msg(msg.OPROC1, 'constructing from poles and zeros');
          pl = plist('gain', varargin{1}, 'poles', varargin{2}, 'zeros', varargin{3}, 'name', varargin{4});
          obj = fromPolesAndZeros(obj, pl);

        otherwise
          [pzs, invars, rest] = utils.helper.collect_objects(args, 'pzmodel');

          %%% Do we have a list of AOs as input
          if ~isempty(pzs) && isempty(rest)
            obj = pzmodel(pzs);
          else
            error('### Unknown number of arguments.');
          end
      end

    end % End constructor

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                Methods (protected)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Access = protected)
    obj = setGain(obj, val)
    obj = setPoles(obj, val)
    obj = setZeros(obj, val)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                               Methods (static)                              %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Static)
```

```
    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: pzmodel.m,v 1.60 2008/09/07 10:48:46 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'pzmodel');
    end

    function out = SETS()
      out = {...
        'Default',        ...
        'From XML File', ...
        'From MAT File', ...
        'From LISO File', ...
        'From Repository', ...
        'From Poles/Zeros', ...
        'From Plist'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
          %--- Pole/zero constructor
        case 'From Poles/Zeros'
          out = plist('gain', 1, ...
            'poles', [], ...
            'zeros', [], ...
            'name', 'None');
          %--- Repository constructor
        case 'From Repository'
          out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
          %--- Read from XML file
        case 'From XML File'
          out = plist('filename', '');
          %--- Read from MAT file
        case 'From MAT File'
          out = plist('filename', '');
          %--- Create from LISO file
        case 'From LISO File'
          out = plist('filename', '');
          %--- Create from a plist
        case 'From Plist'
          out = plist('Plist', []);
        otherwise
          out = plist();
      end
    end % function out = getDefaultPlist(varargin)

  end % End static methods

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                          Methods (static, private)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static, Access=private)

    [ao,bo] = abcascade(a1,b1,a2,b2)

  end % End static, private methods

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                Methods (public)                             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods
    varargout = char(varargin)
```

```
    varargout = display(varargin)
    varargout = string(varargin)
    varargout = copy(varargin)

    f = getlowerFreq(varargin)
    f = getupperFreq(varargin)
    f = tomiir(varargin)
    f = tomfir(varargin)
    varargout = pzm2ab(varargin)
    varargout = fngen(varargin)
    varargout = resp(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                           Methods (protected)                             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Access = protected)
    pzm = fromLISO(pzm, pli)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                           Methods (private)                               %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Access = private)
    % Constructors
    varargout = fromPolesAndZeros(varargin)

    % Converters
    [a,b] = cp2iir(p, fs)
    [a,b] = rp2iir(p, fs)
    [a,b] = cz2iir(z, fs)
    [a,b] = rz2iir(z, fs)
  end

end % End classdef

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

downsample

PURPOSE ^

DOWNSAMPLE AOs containing time-series data.

SYNOPSIS ^

function varargout = downsample(varargin)

DESCRIPTION ^

```
 DOWNSAMPLE AOs containing time-series data.
%%%
```

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = downsample(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Check for input plist
  if isempty(pl)
    error('### Please give a plist with a parameter ''Factor''.');
  end

  % Get parameters from plist
  offset = find(pl, 'offset');
  factor = find(pl, 'factor');

  % Checking downsampling value is valid
  if isempty(factor)
    error('### Please give a plist with a parameter ''Factor''.');
  end

  % Checking downsampling value is integer
  if rem(factor, floor(factor)) ~= 0
    warning('!!! Downsample factor should be an integer. Rounding. !!!');
    factor = round(factor);
  end

  % Checking sample offset value
  if isempty(offset)
    warning('!!! No offset specified; using default of 0 samples !!!');
    offset = 0;
  end

```

```
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = upsample(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % combine plists
  pl = combine(pl, getDefaultPlist());

  % Get output sample rate
  Nup = find(pl, 'N');
  if isempty(Nup)
    error('### Please give a plist with a parameter ''N''.');
  end
  % Get initial phase
  phase = find(pl, 'phase');
  if isempty(phase)
    phase = 0;
  end

  % Loop over AOs
  for j=1:numel(bs)
    if ~isa(bs(j).data, 'tsdata')
      warning('!!! Upsample only works on tsdata objects. Skipping AO %s', ao_invars{j});
      bs(j) = [];
    else
      % upsample y
      bs(j).setY(upsample(bs(j).data.y, floor(Nup), phase), 'internal');
      % Correct fs
      bs(j).setFs(floor(Nup)*bs(j).data.fs, 'internal');
      % Set name
      bs(j).setName(sprintf('upsample(%s)', ao_invars{j}), 'internal');
      % Add history
      bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    end
  end

  % Set outputs
  if nargout > 0
    varargout{1} = bs;
  end
end

%---------------------------------------------------------------------------
% Get Info Object
```

```
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: upsample.m,v 1.12 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('N', 1, 'phase', 0);
end
% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

resample

PURPOSE ^

RESAMPLE overloads resample function for AOs.

SYNOPSIS ^

function varargout = resample(varargin)

DESCRIPTION ^

```
 RESAMPLE overloads resample function for AOs.
%%%
```

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = resample(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  if isempty(pl)
    error('### Please give a plist with a parameter ''fsout''.');
  end

  % Get output sample rate
  fsout = find(pl, 'fsout');
  if isempty(fsout)
    error('### Please give a plist with a parameter ''fsout''.');
  end

  % Get a filter if specified
  filt = find(pl, 'filter');

  % Loop over AOs
  for j=1:numel(bs)
    if ~isa(bs(j).data, 'tsdata')
      warning('!!! Skipping non-tsdata AO: %s', ao_invars{j});
      bs(j) = [];
    else
      % Compute the resampling factors
      [P,Q] = utils.math.intfact(fsout,bs(j).data.fs);
      utils.helper.msg(msg.PROC1, 'resampling by %g/%g', P, Q);

      % Check we have an evenly sampled data series
      [fs, t0, fitted] = tsdata.fitfs(bs(j).data.getX);
      if fitted
        error('### The AO %s is unevenly sampled. It can not be resampled this way.', ao_invars{j});
      end
      % resample y
      if isempty(filt)
        bs(j).setY(resample(bs(j).data.getY, P, Q), 'internal');
      else
        [p,q] = rat( fsout/d.fs, 1e-12 );
        b = p*filt.a;
        bs(j).setY(resample(bs(j).data.getY, P, Q, b), 'internal');
      end
      % Set new sample rate
      bs(j).setFs(fsout, 'internal');
```

```
      % Set output AO name
      bs(j).setName(sprintf('resample(%s)', ao_invars{j}), 'internal');
      % Add history
      bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: resample.m,v 1.30 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('fsout', []);
end
% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = interp(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Get parameters
  vertices = find(pl, 'vertices');
  method   = find(pl, 'method');

  %-----------------------
  % Loop over input AOs
  for j=1:numel(bs)
    %----------------------------
    % Interpolate this vector
    if ~isa(bs(j).data, 'cdata')
      % for tsdata, fsdata and xydata objects
      bs(j).data.setXY(vertices, interp1(bs(j).data.getX,bs(j).data.getY,vertices,
```

```
method, 'extrap'));
    else
      % for cdata object
      bs(j).data.setY(interp1(bs(j).data.y,vertices, method, 'extrap'));
    end

    % Adjust sample rate for tsdata
    if isa(bs(j).data, 'tsdata')
      utils.helper.msg(msg.PROC1, 'adjusting sample rate of new data to best fit');
      [fs, t0, fitted] = tsdata.fitfs(bs(j).data.getX);
      utils.helper.msg(msg.PROC2, 'got new sample rate of %g Hz', fs);
      bs(j).data.setFs(fs);
      if ~fitted
        bs(j).data.collapseX;
      end
    end
    % Add history
    bs(j).addHistory(getInfo, pl, ao_invars, bs(j).hist);
    % set name
    bs(j).setName(sprintf('%s(%s)', method, ao_invars{j}), 'internal');
  end

  % Set outputs
  if nargout > 0
    varargout{1} = bs;
  end
end

%-----------------------------------------------------------------------------
% Get Info Object
%-----------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: interp.m,v 1.24
2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%-----------------------------------------------------------------------------
% Get Default Plist
%-----------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('vertices',  [], 'method', 'spline');
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 AO.

 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 AO.
```

```
  %
  % 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
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout=ltpda_spikecleaning(varargin)
  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/spikecleaning now.');

  % Call ao/spikecleaning
  bs =  spikecleaning(varargin{:});

  % set output
  varargout{1} = bs;

end
```

Generated on Mon 08-Sep-2008 13:18:47 by ***m2html*** *© 2003*

ltpda_gapfilling

PURPOSE ^

LTPDA_GAPFILLING fills possible gaps in data.

SYNOPSIS ^

function varargout=ltpda_gapfilling(varargin)

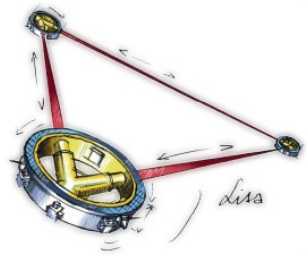
DESCRIPTION ^

```
 LTPDA_GAPFILLING fills possible gaps in data.

%%%
```

```
%                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 datare_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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout=ltpda_gapfilling(varargin)

  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/gapfilling now.');

  % Call ao/gapfilling
  bs =  gapfilling(varargin{:});

  % set output
  varargout{1} = bs;

end
```

Generated on Mon 08-Sep-2008 13:18:47 by ***m2html*** *© 2003*

LISA
GROUP HANNOVER, GERMANY



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 äveraging 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.

WWW.LISA.AEI-HANNOVER.DE

Universität Hannover
Max-Planck-Institut für Gravitationsphysik (Albert-Einstein-Institut)

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 ^

```
% 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
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_pwelch(varargin)

  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/psd now.');

  % Call ao/psd
  bs =  psd(varargin{:});

  % set output
  varargout{1} = bs;

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.
%%%
```

```
VERSION:     $Id: specwin.m,v 1.53 2008/09/03 16:39:05 hewitson Exp $

HISTORY:     30-01-07 M Hewitson
                Creation

%%%
```

- 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 ^

```
%  % 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%       w = specwin('Flattop', N, psll) - create a specwin Flattop window
%                                         with the prescribed psll.

classdef specwin < ltpda_nuo

  %-------------------------------------------------
  %---------- Private read-only Properties --------
  %-------------------------------------------------
  properties (SetAccess = private)
    type     = '';
    alpha    = [];
    psll     = [];
    rov      = [];
    nenbw    = [];
    w3db     = [];
    flatness = [];
    ws       = [];
    ws2      = [];
    win      = [];
  end

  properties (SetAccess = protected)
    version = '$Id: specwin.m,v 1.53 2008/09/03 16:39:05 hewitson Exp $';
  end

  %-------------------------------------------------
  %-------- Declaration of public methods --------
  %-------------------------------------------------
  methods

    %-----------------------------------------------
    %-------- Property rules                 --------
    %-----------------------------------------------

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                   Constructor                                %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function ww = specwin(varargin)

      import utils.const.*
      utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);

      %%% Call superclass
      ww = ww@ltpda_nuo(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%

      switch nargin
        case 0
          utils.helper.msg(msg.OPROC1, 'empty constructor');
        case 1
          if isa(varargin{1}, 'specwin')
            utils.helper.msg(msg.OPROC1, 'copy constructor');
            %%%%%%%%%%   spw = specwin(specwin-object)   %%%%%%%%%%
            % copy existing specwin
            ww = copy(varargin{1}, 1);
          elseif isstruct(varargin{1})
            utils.helper.msg(msg.OPROC1, 'constructing from struct');
            %%%%%%%%%%   spw = specwin(struct)   %%%%%%%%%%

            %%% Set properties which are declared in this class
            spw_struct = varargin{1};

            ww.type     = spw_struct.type;
            ww.alpha    = spw_struct.alpha;
            ww.psll     = spw_struct.psll;
            ww.rov      = spw_struct.rov;
            ww.nenbw    = spw_struct.nenbw;
            ww.w3db     = spw_struct.w3db;
            ww.flatness = spw_struct.flatness;
            ww.ws       = spw_struct.ws;
            ww.ws2      = spw_struct.ws2;
            ww.win      = spw_struct.win;
            ww.version  = spw_struct.version;

          elseif ischar(varargin{1})
            utils.helper.msg(msg.OPROC1, 'constructing from string');
            error('### Code up the char constructor.');
          end
        case 2
            utils.helper.msg(msg.OPROC1, 'constructing type %s', varargin{1});
```

```
          if ischar(varargin{1})
            %%%%%%%%%%   spw = specwin('Win_type', N)   %%%%%%%%%%
            ww = get_window(ww, varargin{1}, varargin{2});
          else
            error('### Unknown 2 argument constructor for specwin object.')
          end
        case 3
          utils.helper.msg(msg.OPROC1, 'constructing type %s', varargin{1});
          %%%%%%%%%%   spw = specwin('Kaiser', N, psll)   %%%%%%%%%%
          ww = get_window(ww, varargin{1}, varargin{2}, varargin{3});
        otherwise
          error('### Unknown number of constructor arguments');
      end % End of constructor

    end

    varargout = copy(varargin)

  end % End public methods


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (Static, Public)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true)

    varargout = update_struct(varargin);

    function ww = getTypes()
      ww = {...
        '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'...
        };
    end

    function out = VEROUT()
      out = '$Id: specwin.m,v 1.53 2008/09/03 16:39:05 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'specwin');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          out = plist();
      end
    end

  end % End static methods

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (Static, Private)                         %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true, Access = private)

    varargout = kaiser_alpha(varargin)
    varargout = kaiser_flatness(varargin)
    varargout = kaiser_nenbw(varargin)
    varargout = kaiser_rov(varargin)
    varargout = kaiser_w3db(varargin)

  end % End static private methods

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                    Methods (protected)                         %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Access = protected)
    varargout = get_window(varargin)
    varargout = win_BH92(varargin)
    varargout = win_Bartlett(varargin)
    varargout = win_FTHP(varargin)
    varargout = win_FTNI(varargin)
```

```
    varargout = win_FTSRS(varargin)
    varargout = win_HFT116D(varargin)
    varargout = win_HFT144D(varargin)
    varargout = win_HFT169D(varargin)
    varargout = win_HFT196D(varargin)
    varargout = win_HFT223D(varargin)
    varargout = win_HFT248D(varargin)
    varargout = win_HFT70(varargin)
    varargout = win_HFT90D(varargin)
    varargout = win_HFT95(varargin)
    varargout = win_Hamming(varargin)
    varargout = win_Hanning(varargin)
    varargout = win_Nuttall3(varargin)
    varargout = win_Nuttall3a(varargin)
    varargout = win_Nuttall3b(varargin)
    varargout = win_Nuttall4(varargin)
    varargout = win_Nuttall4a(varargin)
    varargout = win_Nuttall4b(varargin)
    varargout = win_Nuttall4c(varargin)
    varargout = win_Rectangular(varargin)
    varargout = win_SFT3F(varargin)
    varargout = win_SFT3M(varargin)
    varargout = win_SFT4F(varargin)
    varargout = win_SFT4M(varargin)
    varargout = win_SFT5F(varargin)
    varargout = win_SFT5M(varargin)
    varargout = win_Welch(varargin)
  end

end % End classdef
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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)
```

```
%
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_cpsd(varargin)

  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/cpsd now.');

  % Call ao/cpsd
  bs =  cpsd(varargin{:});

  % set output
  varargout{1} = bs;
end
```

Generated on Mon 08-Sep-2008 13:18:47 by ***m2html*** *© 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_cohere(varargin)
  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/cohere now.');

  % Call ao/cohere
  bs =  cohere(varargin{:});

  % set output
  varargout{1} = bs;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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')
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_tfe(varargin)

  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/tfe now.');


  % Call ao/tfe
  bs =  tfe(varargin{:});

  % set output
  varargout{1} = bs;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

This function calls:

This function is called by:

SOURCE CODE ^

```
% LTPDA_LPSD implement LPSD algorithm for analysis objects.
%%%
```

ltpda_lcpsd

PURPOSE ^

LTPDA_LCPSD implement weighted cross-power estimation computed on a log frequency axis.

SYNOPSIS ^

function varargout = ltpda_lcpsd(varargin)

DESCRIPTION ^

```
 LTPDA_LCPSD implement weighted cross-power estimation computed on a log frequency axis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 LTPDA_LCPSD 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 PSLL]
     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 ^

```
% LTPDA_LCPSD implement weighted cross-power estimation computed on a log frequency axis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% LTPDA_LCPSD 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 PSLL]
%     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 $
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_lcpsd(varargin)

  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/lcpsd now.');

  % Call ao/lcpsd
  bs =  lcpsd(varargin{:});

  % set output
  varargout{1} = bs;
end
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 PSLL]
     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.
%%%
```

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 PSLL]
     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 ^

```
% 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 PSLL]
%     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 $
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_ltfe(varargin)

  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/ltfe now.');

  % Call ao/ltfe
  bs =  ltfe(varargin{:});

  % set output
  varargout{1} = bs;
end
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = polyfit(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Degree of polynomial to fit
  N      = find(pl, 'N');
  coeffs = find(pl, 'coeffs');

  %------------------------
  % Loop over input AOs
  for j=1:numel(bs)
    if isa(bs(j).data, 'cdata')
      warning('!!! Can''t fit to cdata objects. Skipping AO %s', ao_invars{j});
      bs(j) = [];
    else
      % Get x values
      x = bs(j).data.getX;
      % Fit polynomial
      if ~isempty(N)
        [p,s] = polyfit(x, bs(j).data.getY, N);
      else
        p = coeffs;
        s = [];
      end
      % make new values from these coefficients
      y = polyval(p, bs(j).data.getX);
      % Set data
      bs(j).data.setXY(x, y);
      % Set output name
      bs(j).setName(sprintf('polyfit(%s)', ao_invars{j}), 'internal');
      % Set extra processing info
      bs(j).procinfo = plist('coeffs', p, 'S', s);
      % Add history
      bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    end
  end

  % Set outputs
  if nargout > 0
    varargout{1} = bs;
```

```
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: polyfit.m,v 1.18
2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('N', 1, 'coeffs', []);
end

% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %
```

```
%                 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.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_lincom(varargin)
  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/lincom now.');

  % Call ao/lincom
  bs =  lincom(varargin{:});

  % set output
  varargout{1} = bs;
end
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

minus

PURPOSE ^

MINUS implements minus operator for analysis objects.

SYNOPSIS ^

function varargout = minus(varargin)

DESCRIPTION ^

```
 MINUS implements minus operator for analysis objects.
%%%
```

```
% VERSION:     $Id: minus.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY:     01-02-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = minus(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl              = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'minus', '-', pl, getInfo);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: minus.m,v 1.24 2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

mpower

PURPOSE ^

MPOWER implements mpower operator for analysis objects.

SYNOPSIS ^

function varargout = mpower(varargin)

DESCRIPTION ^

```
 MPOWER implements mpower operator for analysis objects.
%%%
```

```
% VERSION:     $Id: mpower.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY:     01-02-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = mpower(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl              = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'mpower', '^', pl, getInfo);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.aop, '$Id: mpower.m,v 1.24 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end
% END
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

mrdivide

PURPOSE ^

MRDIVIDE implements mrdivide operator for analysis objects.

SYNOPSIS ^

function varargout = mrdivide(varargin)

DESCRIPTION ^

```
 MRDIVIDE implements mrdivide operator for analysis objects.
%%%
```

```
% VERSION:     $Id: mrdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY:     01-02-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = mrdivide(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl              = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'mrdivide', '/', pl, getInfo);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: mrdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end
% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

mtimes

PURPOSE ^

MTIMES implements mtimes operator for analysis objects.

SYNOPSIS ^

function varargout = mtimes(varargin)

DESCRIPTION ^

```
 MTIMES implements mtimes operator for analysis objects.
%%%
```

```
% VERSION:     $Id: mtimes.m,v 1.28 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY:     01-02-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = mtimes(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl              = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'mtimes', '*', pl, getInfo);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: mtimes.m,v 1.28 2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end
% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

plus

PURPOSE ^

PLUS implements addition operator for analysis objects.

SYNOPSIS ^

function varargout = plus(varargin)

DESCRIPTION ^

```
 PLUS implements addition operator for analysis objects.
%%%
```

```
% VERSION:     $Id: plus.m,v 1.37 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY: 01-02-07 M Hewitson
%              Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = plus(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl              = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'plus', '+', pl, getInfo);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: plus.m,v 1.37 2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

power

PURPOSE ^

POWER implements power operator for analysis objects.

SYNOPSIS ^

function varargout = power(varargin)

DESCRIPTION ^

```
 POWER implements power operator for analysis objects.
%%%
```

```
% VERSION:     $Id: power.m,v 1.22 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY:     01-02-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = power(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl              = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'power', '.^', pl, getInfo);
end

%----------------------------------------------------------------------------
% Get Info Object
%----------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.aop, '$Id: power.m,v 1.22 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%----------------------------------------------------------------------------
% Get Default Plist
%----------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end

% END
```

Generated on Mon 08-Sep-2008 13:18:47 by ***m2html*** *© 2003*

rdivide

PURPOSE ^

RDIVIDE implements rdivide operator for analysis objects.

SYNOPSIS ^

function varargout = rdivide(varargin)

DESCRIPTION ^

```
 RDIVIDE implements rdivide operator for analysis objects.
%%%
```

```
% VERSION:     $Id: rdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY:     01-02-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = rdivide(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl               = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'rdivide', './', pl, getInfo);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: rdivide.m,v 1.24 2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

times

PURPOSE ^

TIMES implements times operator for analysis objects.

SYNOPSIS ^

function varargout = times(varargin)

DESCRIPTION ^

```
 TIMES implements times operator for analysis objects.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = times(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input ao's, plist's and ao variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Check for numbers in the other arguments and convert to AOs
  args = utils.helper.collect_values(varargin);
  [as, ao_invars] = utils.helper.collect_objects(args, 'ao', in_names);
  pl              = utils.helper.collect_objects(args, 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  varargout{1} = applyoperator(bs, ao_invars, 'times', '.*', pl, getInfo);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.aop, '$Id: times.m,v 1.25
2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end
% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
%               >> 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = attachm(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pl, pl_invars,rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Get filename
  mfile = find(pl, 'filename');
  if isempty(mfile)
    % get filename from 'rest'
    for jj=1:numel(rest)
      if ischar(rest{jj})
        mfile = rest{jj};
        break;
      end
    end
  end
  if isempty(mfile)
    error('### No filename found in inputs.');
  end

  % Loop over AOs
  for jj=1:numel(bs)

    % get file parts
    [pathstr,name,ext,vers] = fileparts(mfile);
    if isempty(ext)
      mfile = [mfile '.m'];
    end
    min = textread(mfile,'%s','delimiter','\n','whitespace','');
    mout = '';
    for j=1:length(min)
      mout = [mout '\n' min{j}];
    end

    % attach to AO
    bs(jj).mfile = mout;
    bs(jj).mfilename = strrep(mfile, '''', '`');
  end

  % Set outputs
  if nargout > 0
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
  end
  % 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.
%%%
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = attachmdl(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  as = utils.helper.collect_objects(varargin(:), 'ao', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % get the current mdl file
  mdlfile = bdroot(gcb);

  % Check its status
  dirty = get_param(bdroot,'Dirty');
  if strcmp(dirty, 'on')
    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');
  end

  % read it
  [pathstr,name,ext] = fileparts(mdlfile);
  if isempty(ext)
    mdlfile = [mdlfile '.mdl'];
  end

  mdlfilein = textread(mdlfile,'%s','delimiter','\n','whitespace','');
  mdlout = '';
  for j=1:length(mdlfilein)
    mdlout = [mdlout '\n' mdlfilein{j}];
  end

  % Now attach it to each input AO
  for jj=1:numel(bs)
    bs(jj).mdlfile = strrep(mdlout, '''', '`');
    bs(jj).mdlfilename = mdlfile;
  end

  % Set outputs
  if nargout > 0
    varargout{1} = bs;
  end
end

%---------------------------------------------------------------------------
% Get Info Object
%---------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: attachmdl.m,v 1.14 2008/09/05 11:05:28 ingo Exp $', sets, pl);
end

%---------------------------------------------------------------------------
% Get Default Plist
%---------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

cat

PURPOSE ^

CAT concatenate AOs into a vector.

SYNOPSIS ^

function varargout = cat(varargin)

DESCRIPTION ^

```
 CAT concatenate AOs into a vector.
%%%
```

```
% VERSION:     $Id: cat.m,v 1.14 2008/09/05 11:05:28 ingo Exp $
%
% HISTORY:     27-03-07 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = cat(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  as  = utils.helper.collect_objects(varargin(:), 'ao', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Set output
  if nargout == 0
    error('### cat cannot be used as a modifier. Please give an output variable.');
  else
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: cat.m,v 1.14 2008/09/05 11:05:28 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

- 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.
- lxspec 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
- xspec XSPEC performs cross-spectral analysis of various forms.
- zeropad ZEROPAD zero pads the input data series.

SOURCE CODE ^

```
% COPY makes a (deep) copy of the input AOs.
%%%
```

```
%%%
```

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.
%%%
```

```
%                  Creation
%
%%%
```

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.
- lxspec 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 TRANSPOSE 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 ^

```
% FIND particular samples that satisfy the input query and return a new AO.
%%%
```

```
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   varargout{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 %--------------------------------------------------------------------------
```

```
% Get Default Plist
%-----------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('query',  '');
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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')
```

```
%
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function varargout = index(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  if nargout == 0
    error('### index cannot be used as a modifier. Please give an output variable.');
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % We can only effectively handle a single matrix of AOs
  af = 0;
  for j=1:nargin
    if isa(varargin{j}, 'ao')
      af = af + 1;
    end
  end
  if af > 1
    error('### Input a single matrix of AOs for indexing.');
  end

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pl, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Get indices
  idxi  = find(pl, 'i');
  idxj  = find(pl, 'j');
  if isempty(idxi) || isempty(idxj)
    % go through the other inputs
    vals  = [];
    for j=1:numel(rest)
      if isnumeric(rest{j}) && numel(rest{j})==1
        vals = [vals rest{j}];
      end
    end
    if ~isempty(vals)
      idxi = vals(1);
      if numel(vals) > 1
        idxj = vals(2);
      end
    end
  end

  if isempty(idxi) && isempty(idxj)
    error('### Please provide an index of index pair.');
  end

  % Gather the input histories
  inhists = [];
  for j=1:numel(bs)
    inhists = [inhists bs(j).hist];
  end

  % Now index with either (i) or (i,j)
  if isempty(idxj)
    aout = bs(idxi);
    inhists = inhists(idxi);
  else
```

```
    aout = bs(idxi, idxj);
    inhists = inhists(sub2ind(size(bs), idxi, idxj));
  end

  % Make sure we store the indices
  pl = pl.pset('i', idxi);
  pl = pl.pset('j', idxj);

  % Name and add history to all outputs
  for j=1:numel(aout)
    if isempty(idxj)
      aout(j).setName(sprintf('%s(%d)', ao_invars{1}, idxi), 'internal');
    else
      aout(j).setName(sprintf('%s(%d,%d)', ao_invars{1}, idxi, idxj), 'internal');
    end
    % add history
    aout(j).addHistory(getInfo, pl, ao_invars, inhists(j));
  end

  % Set output
  varargout{1} = aout;
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: index.m,v 1.17
2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('i',  [], 'j', []);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

join

PURPOSE ^

JOIN multiple AOs into a single AO.

SYNOPSIS ^

function varargout = join(varargin)

DESCRIPTION ^

```
 JOIN multiple AOs into a single AO.
%%%
```

SOURCE CODE ^

```
% JOIN multiple AOs into a single AO.
%%%
```

```
            yo = as(j).data.getY;
            xo = x;
          else
            idxPost = find(x > max(xo));
            idxPre  = find(x < min(xo));
            xo = [x(idxPre); xo; x(idxPost)];
            yo = [as(j).data.getY(idxPre); yo; as(j).data.getY(idxPost)];
          end

          % store fs
          if fs>0 && as(j).data.fs ~= fs
            error('!!! Data has different sample rates !!!');
          end
          fs = as(j).data.fs;

          % store xunits
          if xunits ~= as(j).data.xunits
            error('!!! Data has different X units !!!');
          end
          xunits = as(j).data.xunits;

          % store yunits
          if yunits ~= as(j).data.yunits
            error('!!! Data has different Y units !!!');
          end
          yunits = as(j).data.yunits;

        case 'fsdata'
          error('### I only work with time-series data at the moment.');
        case 'xydata'
          error('### I only work with time-series data at the moment.');
        case 'cdata'
          error('### I only work with time-series data at the moment.');
        otherwise
          error('### Unknown data type');
      end
      % Collect this input history
      histin = [histin as(j).hist];
      % Collect names, invars
      if ~isempty(aname)
        if ~strcmp(aname, as(j).name)
          aname = [aname ',' as(j).name];
        end
      else
        aname = as(j).name;
      end
    else
      warning('!!! Ignoring AO input with data type %s', dtype);
    end
  end
  % Tidy up names
  dname = dname(2:end);

  %-----------------------------------------------
  % Now sort output vectors
  [xos, idx] = sort(xo);
  yos = yo(idx);

  %% Build output data object
  switch dtype
    case 'tsdata'
      % get t0
      t0  = xos(1);
      xos = xos - t0;
      data = tsdata(xos, yos, fs);
      data.setT0((Toff+t0*1000));
      data.setXunits(xunits);
      data.setYunits(yunits);
    case 'fsdata'
    case 'xydata'
    case 'cdata'
  end

  %-----------------------------------------------
  % Build output AO
  a = ao(data);
  a.setName(aname, 'internal');
  % add history
  a.addHistory(getInfo, plist, ao_invars, histin);

  %-----------------------------------------------
  % set output
  varargout{1} = a;
end

%-------------------------------------------------------------------------
```

```
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: join.m,v 1.24 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist();
end

%--------------------------------------------------------------------------
% Get Offset of this set of time-vectors
%--------------------------------------------------------------------------
function Toff = getToff(as)
  Toff = 1e20;
  for j=1:length(as)
    if isa(as(j).data, 'tsdata')
      t0 = as(j).data.t0.utc_epoch_milli;
      if t0 < Toff
        Toff = t0;
      end
    end
  end
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
- lxspec LXSPEC performs log-scale cross-spectral analysis of various forms.
- xcorr XCORR makes cross-correlation estimates of the time-series
- xspec XSPEC performs cross-spectral analysis of various forms.

SUBFUNCTIONS ^

- function ii = getInfo(varargin)
- function pl_default = getDefaultPlist()

SOURCE CODE ^

```
% 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
%
%%%
```

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.
%%%
```

```
%
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = md5(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  as = utils.helper.collect_objects(varargin(:), 'ao', in_names);

  h = {};
  for ii = 1:numel(as)

    %%%%%%%%%%% Convert object to XML
    % make pointer to xml document
    xml = com.mathworks.xml.XMLUtils.createDocument('ltpda_object');
    % extract parent node
    parent = xml.getDocumentElement;
    % write obj into xml
    utils.helper.xmlwrite(as(ii), xml, parent, '');    % Save the XML document.

    h = [h cellstr(utils.prog.hash(xmlwrite(xml), 'MD5'))];

  end

  % Set outputs
  h = reshape(h, size(as));
  if numel(h) == 1
    varargout{1} = cell2mat(h);
  else
    varargout{1} = h;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: md5.m,v 1.12
2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist();
end

```

search

PURPOSE ^

SEARCH selects AOs that match the given name.

SYNOPSIS ^

function varargout = search(varargin)

DESCRIPTION ^

```
 SEARCH selects AOs that match the given name.
%%%
```

```
%               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
%
%%%
```

```
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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setDescription(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Internal call: Only one object + don't look for a plist
  if strcmp(varargin{end}, 'internal')

    %%% decide whether we modify the first object, or create a new one.
    varargin{1} = copy(varargin{1}, nargout);

    varargin{1}.description = varargin{2};
    varargout{1} = varargin{1};
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');

  %%% If pls contains only one plist with the only key 'description' then set the
  %%% property with a plist.
  if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'description')
    rest{1} = find(pls, 'description');
  end

  if numel(rest) ~= 1
    error('### Please specify a value for description, either in a plist or directly.');
  end

  %%% Combine plists
  if isempty(pls)
    pls = plist('description', rest{1});
  else
    pls = pls.combine(plist('description', rest{1}));
  end

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over AOs
  for j=1:numel(bs)
    bs(j).description = rest{1};
    bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  else
    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.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setFs(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Internal call: Only one object + don't look for a plist
  if strcmp(varargin{end}, 'internal')

    %%% decide whether we modify the first object, or create a new one.
    varargin{1} = copy(varargin{1}, nargout);

    varargin{1}.data.setFs(varargin{2});
    varargout{1} = varargin{1};
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');

  %%% If pls contains only one plist with the only key 'fs' then set the
  %%% property with a plist.
  if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'fs')
    rest{1} = find(pls, 'fs');
  end

  if numel(rest) ~= 1
    error('### Please specify a value for Fs, either in a plist or directly.');
  end

  %%% Combine plists
  if isempty(pls)
    pls = plist('fs', rest{1});
  else
    pls = pls.combine(plist('fs', rest{1}));
  end

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over AOs
  for j=1:numel(bs)
    bs(j).data.setFs(rest{1});
    bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

  %--------------------------------------------------------------------------
  % Get Info Object
  %--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: setFs.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
```

```
end

  %--------------------------------------------------------------------------
  % Get Default Plist
  %--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('fs', NaN);
  % END
end

```

*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.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setT0(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Internal call: Only one object + don't look for a plist
  if strcmp(varargin{end}, 'internal')

    %%% decide whether we modify the first object, or create a new one.
    varargin{1} = copy(varargin{1}, nargout);

    varargin{1}.data.setT0(varargin{2});
    varargout{1} = varargin{1};
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');

  %%% If pls contains only one plist with the only key 't0' then set the
  %%% property with a plist.
  if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 't0')
    rest{1} = find(pls, 't0');
  end

  if numel(rest) ~= 1
    error('### Please specify a value for t0, either in a plist or directly.');
  end

  %%% Combine plists
  if isempty(pls)
    pls = plist('t0', rest{1});
  else
    pls = pls.combine(plist('t0', rest{1}));
  end

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over AOs
  for j=1:numel(bs)
    bs(j).data.setT0(rest{1});
    bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
  end
```

```
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.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setX(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Internal call: Only one object + don't look for a plist
  if strcmp(varargin{end}, 'internal')

    %%% decide whether we modify the first object, or create a new one.
    varargin{1} = copy(varargin{1}, nargout);

    varargin{1}.data.setX(varargin{2});
    varargout{1} = varargin{1};
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');

  %%% If pls contains only one plist with the only key 'x' then set the
  %%% property with a plist.
  if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'x')
    rest{1} = find(pls, 'x');
  end

  if numel(rest) ~= 1
    error('### Please specify a value for x, either in a plist or directly.');
  end

  %%% Combine plists
  if isempty(pls)
    pls = plist('x', rest{1});
  else
    pls = pls.combine(plist('x', rest{1}));
  end

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over AOs
  for j=1:numel(bs)
    bs(j).data.setX(rest{1});
    bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: setX.m,v 1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('x', []);
end

```

*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 ^

```
% 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
%
%%%
```

```
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, 'ao', '', utils.const.categories.helper, '$Id: setXY.m,v 1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('x', [], 'y', []);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setXunits

PURPOSE ^

SETXUNITS sets the 'xunits' property of the ao.

SYNOPSIS ^

function varargout = setXunits(varargin)

DESCRIPTION ^

```
 SETXUNITS sets the 'xunits' property of the ao.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setXunits(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Internal call: Only one object + don't look for a plist
  if strcmp(varargin{end}, 'internal')

    %%% decide whether we modify the first object, or create a new one.
    varargin{1} = copy(varargin{1}, nargout);

    varargin{1}.data.setXunits(varargin{2});
    varargout{1} = varargin{1};
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');

  %%% If pls contains only one plist with the only key 'xunits' then set the
  %%% property with a plist.
  if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'xunits')
    rest{1} = find(pls, 'xunits');
  end

  if numel(rest) ~= 1
    error('### Please specify a value for xunits, either in a plist or directly.');
  end

  %%% Combine plists
  if isempty(pls)
    pls = plist('xunits', rest{1});
  else
    pls = pls.combine(plist('xunits', rest{1}));
  end

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over AOs
  for j=1:numel(bs)
    bs(j).data.setXunits(rest{1});
    bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
```

```
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.
%%%
```

```
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 %%%%
```

```
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.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setYunits(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Internal call: Only one object + don't look for a plist
  if strcmp(varargin{end}, 'internal')

    %%% decide whether we modify the first object, or create a new one.
    varargin{1} = copy(varargin{1}, nargout);

    varargin{1}.data.setYunits(varargin{2});
    varargout{1} = varargin{1};
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');

  %%% If pls contains only one plist with the only key 'yunits' then set the
  %%% property with a plist.
  if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'yunits')
    rest{1} = find(pls, 'yunits');
  end

  if numel(rest) ~= 1
    error('### Please specify a value for yunits, either in a plist or directly.');
  end

  %%% Combine plists
  if isempty(pls)
    pls = plist('yunits', rest{1});
  else
    pls = pls.combine(plist('yunits', rest{1}));
  end

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over AOs
  for j=1:numel(bs)
    bs(j).data.setYunits(rest{1});
    bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)

  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
```

```
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setZ

PURPOSE ^

SETZ sets the 'z' property of the ao.

SYNOPSIS ^

function varargout = setZ(varargin)

DESCRIPTION ^

```
 SETZ sets the 'z' property of the ao.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setZ(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Internal call: Only one object + don't look for a plist
  if strcmp(varargin{end}, 'internal')

    %%% decide whether we modify the first object, or create a new one.
    varargin{1} = copy(varargin{1}, nargout);

    varargin{1}.data.setZ(varargin{2});
    varargout{1} = varargin{1};
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars,rest] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pls,  invars, rest] = utils.helper.collect_objects(rest(:), 'plist');

  %%% If pls contains only one plist with the only key 'z' then set the
  %%% property with a plist.
  if length(pls) == 1 && isa(pls, 'plist') && nparams(pls) == 1 && isparam(pls, 'z')
    rest{1} = find(pls, 'z');
  end

  if numel(rest) ~= 1
    error('### Please specify a value for z, either in a plist or directly.');
  end

  %%% Combine plists
  if isempty(pls)
    pls = plist('z', rest{1});
  else
    pls = pls.combine(plist('z', rest{1}));
  end

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over AOs
  for j=1:numel(bs)
    bs(j).data.setZ(rest{1});
    bs(j).addHistory(getInfo, pls, ao_invars(j), bs(j).hist);
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.helper, '$Id: setZ.m,v 1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('z', '');
end

```

*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
object(s).
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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:
```

```
%               >> 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
%
%%%
```

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 %
```

```
% VERSION:     $Id: timeshift.m,v 1.11 2008/09/05 11:05:29 ingo Exp $
%
% HISTORY:     23-02-08 M Hewitson
%                 Creation
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
                matches(jj) = 1;
              end
            end
          catch
            warning('!!! failed to test against: %s', mfile);
          end
        end % End loop over files on MATLAB path

        if matches(jj)==0
          fails = which(fcnname, '-ALL');
          for ff=1:length(fails)
            utils.helper.msg(msg.PROC1, 'Illegal function: %s', fails{ff});
          end
        end
      end % end loop over nodes

      % Decide whether or not this AO is valid
      if sum(matches) == length(nodes)
        passed(ec) = 1;
        utils.helper.msg(msg.PROC1, 'AO validated');
      else
        passed(ec) = 0;
        utils.helper.msg(msg.PROC1, 'AO not validated');
      end
    end % end loop over all objects

    % Set outputs
    varargout{1} = passed;
  end

  %--------------------------------------------------------------------------
  % Get Info Object
  %--------------------------------------------------------------------------
  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, 'ao', '', utils.const.categories.helper, '$Id: validate.m,v 1.6
2008/09/05 11:05:29 ingo Exp $', sets, pl);
  end

  %--------------------------------------------------------------------------
  % Get Default Plist
  %--------------------------------------------------------------------------
  function pl_default = getDefaultPlist()
    pl_default = plist();
  end
  % END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
%
%               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
%
%%%
```

```
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 %%%%
```

```
%%%
```

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.
%%%
```

```
%
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = mdc1_ifo2acc_fd(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);
  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % combine plists
  pl = combine(pl, getDefaultPlist());
  %% Extract parameters from plist

  % Get parameters out
  w1     = find(pl, 'Omega1');
  w3     = find(pl, 'Omega3');
  delta  = find(pl, 'delta');
  o1xx     = bs(1);
  o12xx    = bs(2);
  o112xx    = bs(3);
  if ~strcmp(o1xx.name, 'o1xx')
    warning('!!! The name of the first input AO is no ''o1''. Are you sure this is the
o1xx frequency-series?');
  end
  if ~strcmp(o12xx.name, 'o12xx')
    warning('!!! The name of the second input AO is no ''o12''. Are you sure this is
```

```
the o12xx frequency-series?');
  end

  if isempty(o1xx.data) || isempty(o12xx.data) || isempty(o112xx.data)
    error('### Please provide two input frequency-series.');
  end
  if ~isequal(o1.data.x, o12.data.x, o112xx.data.x)
    error('### The three input frequency-series should be computed at the same frequencies.');
  end

  % Get frequency vector from one of the input spectra
  f = o1xx.data.getX;

  %% Compute response for frequencies f

  [a11xx, a22xx] = computeAcc(f, w1, w3, delta, o1xx, o12xx, o112xx);

  varargout{1} = a11xx;
  varargout{2} = a22xx;
end
%-------------------------------------------------------------------------
% Get DF controller for each frequency
function [a11xx, a22xx] = computeAcc(f, w1, w3, delta, o1xx, o12xx, o112xx)

  % A model of the downsampling from 100 to 10Hz
  pl = plist('gain',1, 'poles', [pz(10), pz(10), pz(10)], 'zeros', []);
  pzm = pzmodel(pl);
  ds  = resp(pzm, plist('f', f));

  % Drag-free
  Cdf  = ltpda_mdc1_C(plist('f', f, 'Controller', 'df'));
  Adf  = ltpda_mdc1_actuator(plist('f', f, 'Actuator', 'df'));
  Cdf  = ds.*Cdf.*Adf;
  Sw1  = ltpda_mdc1_dynamics(plist('f', f, 'Omega2', w1.data.y.^2));
  Sw1.setName('Sw1', 'internal');

  % Suspension
  Csus = ltpda_mdc1_C(plist('f', f, 'Controller', 'sus'));
  Asus = ltpda_mdc1_actuator(plist('f', f, 'Actuator', 'sus'));
  Csus = ds.*Csus.*Asus;
  Sw3  = ltpda_mdc1_dynamics(plist('f', f, 'Omega2', w3.data.y.^2));
  Sw3.setName('Sw3', 'internal');

  % Square terms for dealing with PSDs
  Csusxx = abs(Csus).^2;
  Cdfxx  = abs(Cdf).^2;
  Sw1xx  = abs(Sw1).^2;
  Sw3xx  = abs(Sw3).^2;

  wd   = w3.^2-w1.^2;
  beta = wd - delta.*Sw3;
  bxx  = abs(beta).^2;

  % Calibrate back to a1

  a11xx = o1xx .* (Sw1xx + Cdfxx - Cdf.*conj(Sw1) - conj(Cdf).*Sw1);
  a11xx.setName('PSD(a11)', 'internal');

  % Calibrate back to a2

  e1 = o1xx.*bxx;
  e2 = sqrt(o12xx).*Sw3;
  e3  = sqrt(o12xx).*Csus;
  a22t1 = e1 + abs(e2 - e3).^2;

  a22t2 = beta.*(conj(Sw3)+conj(Csus)).*o112xx;
  a22t3 = conj(beta).*(Sw3+Csus).*o112xx;
  a22xx =  abs(a22t1 ...
    + a22t2  ...
    + a22t3);
  a22xx.setName('PSD(a22)', 'internal');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 Local Functions                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getInfo
%
% DESCRIPTION: Get Info Object
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
```

```
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = mdc1_ifo2acc_inloop(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % combine plists
  pl = combine(pl, getDefaultPlist());

  %% Extract parameters from plist

  % Get parameters out
  w1     = find(pl, 'Omega1');
  w3     = find(pl, 'Omega3');
  delta  = find(pl, 'delta');
  o1     = bs(1);
  o12    = bs(2);

  if ~strcmp(o1.name, 'o1')
    warning('!!! The name of the first input AO is not ''o1''. Are you sure this is the o1 time-series?');
  end
  if ~strcmp(o12.name, 'o12')
    warning('!!! The name of the second input AO is not ''o12''. Are you sure this is the o1 time-series?');
  end

  if isempty(o1.data) || isempty(o12.data)
    error('### Please provide two input time-series.');
  end
  if ~isequal(o1.data.x, o12.data.x)
    error('### The two input time-series should be computed at the same times.');
  end


  %% Compute response for frequencies f

  [a1, a2] = computeAcc(w1, w3, delta, o1, o12, ao_invars);
```

```
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%
```

```
% pole 300.6237575544m 968.3134470825m  ### fitted (name = pole2)
% zero 5.8794500998  ### fitted (name = zero2)
% zero 40.1685309724  ### fitted (name = zero3)
% factor 1

function o1 = applyCdf(o1)

  % Sample frequency filters are designed for
  filtfs = 100;

  if o1.data.fs ~= filtfs
    warning('### Filters are designed for %dHz. Resampling time-series.', filtfs);
    fsin = o1.data.fs;
    o1.resample(plist('fsout', filtfs));
  end

  %---- First filter stage
  a = [27801.38587094533 -107218.527103749 180226.5071750752 ...
    -173891.2206875895 101664.0906613848 -33271.76258436936 4689.527541600029];
  b = [1 -3.833870654386819 6.42742334310825 -6.197734830947992 ...
    3.622482927835305 -1.185355897309293 0.1670551117005498];
  ff1 = miir(a,b,100);

  %---- Second filter stage
  a = [4.312632196506008e-05 -0.0001281084965932162 0.0001404024385649956 ...
    -6.89671676763067e-05 1.354691430208644e-05];
  b = [1 -2.971420266964361 3.257128504882159 -1.599992417674869 0.3142841797676335];
  ff2 = miir(a,b,100);

  %---- Third filter stage
  a = [0.00095273516542127803953  -0.000919529723213437797 17  5.5197001381098205063e-05
8.7265653018697098768e-05];
  b = [1 -2.4840746702182161521  1.9782850928995776574 -0.49403473523860202965];
  ff3 = miir(a,b,100);

  % Apply filters
  o1.filter(ff1);
  o1.filter(ff2);
  o1.filter(ff3);

  % Apply delay filter
  o1.delay(plist('N', 31));

  % Apply actuator
  pzm = pzmodel(1, pz(0.8), []);
  af  = miir(pzm, plist('fs', filtfs));
  o1.filter(af);

  % Resample if necessary
  if fsin ~= filtfs
    o1.resample(plist('fsout', fsin));
  end
end

%-----------------------------------------------------------------------------
% Apply suspension controller designed in LISO
%
% pole 13.5111750971m 765.5054182119m  ### fitted (name = pole1)
% zero 1.1290293787m 906.1738368029m  ### fitted (name = zero0)
% factor 1  ### fitted
%
% pole 0.0000099821f  ### fitted (name = pole0)
% pole 17.3189756179m  ### fitted (name = pole2)
% zero 10.9038956793  ### fitted (name = zero1)
% delay 27.2832312829m  ### fitted
% factor 17126.3998612698e9
%
function o12 = applyCsus(o12)

  % Sample frequency filters are designed for
  filtfs = 100;

  if o12.data.fs ~= filtfs
    warning('### Filters are designed for %dHz. Resampling time-series.', filtfs);
    fsin = o12.data.fs;
    o12.resample(plist('fsout', filtfs));
  end

  %---- First filter stage
  a = [142.9896507914776862  -285.96810764145169514 142.97845756946250617];
  b = [1 -1.9988914209045689496 0.99889214047646490702];
  ff1 = miir(a,b,100);

  %---- Second filter stage
  a = [2.146238155432156e-11 -2.556622712681001e-11 1.278311356340501e-11 -
2.840691902978891e-12];
```

```
  b = [1 -2.499139379998971 1.998822309472276 -0.4996829294733051];
  ff2 = miir(a,b,100);

  % Apply filters
  o12.filter(ff1);
  o12.filter(ff2);

  % Apply delay filter
  o12.delay(plist('N', 30));

  % Apply actuator
  pzm = pzmodel(1, pz(8), []);
  af  = miir(pzm, plist('fs', filtfs));
  o12.filter(af);

  % Resample if necessary
  if fsin ~= filtfs
    o12.resample(plist('fsout', fsin));
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.mdc01, '$Id:
mdc1_ifo2control.m,v 1.8 2008/09/05 14:15:14 hewitson Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
%%%
```

```
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.
%%%
```

SOURCE CODE ^

```
% ABS overloads the Absolute value method for Analysis objects.
%%%
```

```
%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl_default = getDefaultPlist()

  pl_default = plist('axis',  'y');
end

```

*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.
%%%
```

```
%               >> 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
%
%%%
```

```
    if bs(1).data.yunits == bs(2).data.yunits
      bs(1).setYunits(bs(1).data.yunits, 'internal');
    else
      error('### Can''t combine data with different units');
    end
  end

  % Add history
  bs(1).addHistory(getInfo, getDefaultPlist, ao_invars, [bs(1).hist bs(2).hist]);

  % Set name
  bs(1).setName(sprintf('complex(%s, %s)', ao_invars{1}, ao_invars{2}), 'internal');

  % Set output
  varargout{1} = bs(1);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.op, '$Id: complex.m,v 1.22 2008/09/05 14:13:27 hewitson Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist();
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

conj

PURPOSE ^

CONJ overloads the conjugate operator for Analysis objects.

SYNOPSIS ^

function varargout = conj(varargin)

DESCRIPTION ^

```
 CONJ overloads the conjugate operator for Analysis objects.
%%%
```

```
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 %%%%
```

ctranspose

PURPOSE ^

CTRANSPOSE overloads the ' operator for Analysis Objects.

SYNOPSIS ^

function varargout = ctranspose(varargin)

DESCRIPTION ^

```
 CTRANSPOSE overloads the ' operator for Analysis Objects.
%%%
```

```
0011 %
0012 % M-FILE INFO: Get information about this methods by calling
0013 %               >> ao.getInfo('ctranspose')
0014 %
0015 %               Get information about a specified set-plist by calling:
0016 %               >> ao.getInfo('ctranspose', 'None')
0017 %
0018 % HISTORY: 08-05-07 A Monsky
0019 %              Creation
0020 %
0021 %%%%
```

```
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.op, '$Id: ctranspose.m,v 1.25 2008/09/05 11:15:19 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('axis', 'xy');
end

```

*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.
%%%
```

```
%               >> 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
%
%%%
```

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 ^

```
% 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
%
%%%
```

*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.
%%%
```

```
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 submited
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 %%%%
```

```
    pl   = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: eig.m,v 1.26 2008/09/05 11:15:19 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('option', '');
end
```

*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 ^

```
% EXP overloads the exp operator for Analysis objects. Exponential.
%%%
```

```
    sets = {};
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: exp.m,v 1.23 2008/09/05 14:11:47 hewitson Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('axis', 'y');
end

```

*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.
%%%
```

```
%               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
%
%%%
```

inv

PURPOSE ^

INV overloads the inverse function for Analysis Objects.

SYNOPSIS ^

function varargout = inv(varargin)

DESCRIPTION ^

```
 INV overloads the inverse function for Analysis Objects.
%%%
```

```
% HISTORY:     08-05-07 A Monsky
%                 Creation
%
%%%
```

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.
%%%
```

```
%               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
%
%%%
```

*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.
%%%
```

```
%               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
%
%%%
```

*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 ^

```
% 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
%
%%%
```

```
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('axis', 'y');
end

```

*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.
%%%
```

```
0002 %%%%
```

```
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.op, '$Id: max.m,v 1.13
2008/09/08 08:31:56 hewitson Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()

  pl_default = plist('axis',  'y');
end

% 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.
%%%
```

```
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 %%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
%
%               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
%
%%%
```

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.
%%%
```

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = min(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Get default parameters
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Apply method to all AOs
  for j=1:numel(bs)
    % get min value
    if isfield(bs(j).data, 'x')
      switch find(pl, 'axis')
        case 'x'
          [mx, idx] = min(bs(j).data.getX);
          my = bs(j).data.getY(idx);
        case 'y'
          [my, idx] = min(bs(j).data.getY);
          mx = bs(j).data.getX(idx);
      end
      bs(j).setXY(mx, my, 'internal');
    else
      [my, idx] = min(bs(j).data.getY);
      bs(j).setY(my, 'internal');
    end
    % remove fs and nsecs
    if ismethod(bs(j).data, 'setFs')
      bs(j).data.setFs(NaN, 'internal');
    end
    if ismethod(bs(j).data, 'setNsecs');
      bs(j).data.setNsecs(NaN, 'internal');
    end
    % append history
    bs(j).addHistory(getInfo('None'), pl, ao_invars(j), bs(j).hist);
    % Set new AO name
    bs(j).setName(['min(' ao_invars{j} ')'], 'internal');
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.op, '$Id: min.m,v 1.14 2008/09/08 08:31:56 hewitson Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()

  pl_default = plist('axis',  'y');
end

% 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.
%%%
```

```
%
%               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
%
%%%
```

norm

PURPOSE ^

NORM overloads the norm operator for Analysis Objects.

SYNOPSIS ^

function varargout = norm(varargin)

DESCRIPTION ^

```
 NORM overloads the norm operator for Analysis Objects.
%%%
```

```
% 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 submited  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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = norm(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % go through analysis objects
  for j=1:numel(bs)
    switch class(bs(j).data)
      case 'cdata'
        % Apply norm method to data
        applymethod(bs(j), ao_invars(j), 'norm', pl, getDefaultPlist, getInfo);
      otherwise
        error('### this function works for cdata type AO only')
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
  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.
%%%
```

```
%
% 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
%
%%%
```

real

PURPOSE ^

REAL overloads the real operator for Analysis objects.

SYNOPSIS ^

function varargout = real(varargin)

DESCRIPTION ^

```
 REAL overloads the real operator for Analysis objects.
%%%
```

```
%               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
%
%%%
```

sign

PURPOSE ^

SIGN overloads the sign operator for Analysis objects.%

SYNOPSIS ^

function varargout = sign (varargin)

DESCRIPTION ^

```
 SIGN overloads the sign operator for Analysis objects.%
%%%
```

```
% 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
%
%%%
```

sort

PURPOSE ^

SORT the values in the AO.

SYNOPSIS ^

function varargout = sort (varargin)

DESCRIPTION ^

```
 SORT the values in the AO.
%%%
```

SOURCE CODE ^

```
% SORT the values in the AO.
%%%
```

```
end

%-----------------------------------------------------------------------------
% Get Info Object
%-----------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.op, '$Id: sort.m,v 1.10 2008/09/05 11:15:19 ingo Exp $', sets, pl);
end

%-----------------------------------------------------------------------------
% Get Default Plist
%-----------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('dim', 'y', 'dir', 'ascend');
end

% 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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = sqrt(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Get default parameters
  pl = combine(pl, getDefaultPlist);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Apply method to all AOs
  applymethod(bs, ao_invars, 'sqrt', pl, getDefaultPlist, getInfo);

  % Set units
  for ii =1:numel(bs)
    app_axis = pl.find('axis');
    if ~isempty(find('X'==upper(app_axis), 1))
      bs(ii).setXunits(feval('sqrt', bs(ii).data.xunits), 'internal');
    end
    if ~isempty(find('Y'==upper(app_axis), 1))
      bs(ii).setYunits(feval('sqrt', bs(ii).data.yunits), 'internal');
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.op, '$Id: sqrt.m,v 1.24 2008/09/05 11:15:19 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()

  pl_default = plist('axis',  'y');
end

% 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.
%%%
```

```
%
% 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
%
%%%
```

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.
%%%
```

```
%%%
```

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

 OTPUTS:      like matlab fct

 PARAMETERS:  'option' - a string or value that can be submited  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 ^

```
% SVD overloads the determinant function for Analysis objects.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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 % OTPUTS:       like matlab fct
0013 %
0014 % PARAMETERS:  'option' - a string or value that can be submited  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 %%%%
```

```
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 ^

TRANSPOSE overloads the .' operator for Analysis Objects.

SYNOPSIS ^

function varargout = transpose(varargin)

DESCRIPTION ^

```
 TRANSPOSE overloads the .' operator for Analysis Objects.
%%%
```

```
%               >> 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
%
%%%
```

```
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.op, '$Id: transpose.m,v 1.22 2008/09/05 11:15:19 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('axis', 'xy');
end

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

uminus

PURPOSE ^

UMINUS overloads the uminus operator for Analysis objects.

SYNOPSIS ^

function varargout = uminus (varargin)

DESCRIPTION ^

```
 UMINUS overloads the uminus operator for Analysis objects.
%%%
```

```
%
%               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
%
%%%
```

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.
%%%
```

```
%
%               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
%
%%%
```

zeropad

PURPOSE ^

ZEROPAD zero pads the input data series.

SYNOPSIS ^

function varargout = zeropad(varargin)

DESCRIPTION ^

```
 ZEROPAD zero pads the input data series.
%%%
```

```
%              '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
%
%%%
```

```
end

%-----------------------------------------------------------------------------
% Get Info Object
%-----------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.op, '$Id: zeropad.m,v 1.11 2008/09/05 11:15:19 ingo Exp $', sets, pl);
end

%-----------------------------------------------------------------------------
% Get Default Plist
%-----------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('Factor', 2);
end
% 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.
%%%
```

```
% 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
%
%%%
```

display

PURPOSE ^

DISPLAY implement terminal display for analysis object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY implement terminal display for analysis object.
%%%
```

```
%               >> 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
%
%%%
```

```
      disp(txt{ii});
    end
  else
    varargout{1} = txt;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.output, '$Id: display.m,v 1.30 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist();
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 of export

```
%               >> 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
%
%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
```

```
%%%
```

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 %
```

```
%%%
```

iplot

PURPOSE ^

IPLOT provides an intelligent plotting tool for LTPDA.

SYNOPSIS ^

function varargout = iplot(varargin)

DESCRIPTION ^

```
 IPLOT provides an intelligent plotting tool for LTPDA.
%%%
```

```
                          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

%%%
```

- 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 ^

```
% IPLOT provides an intelligent plotting tool for LTPDA.
%%%
```

```
%             '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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% TODO:
%    1) Add XRange, YRange, ZRange to xyzdata
%    5) How do we deal with axis units? Symbolic math toolbox?
%    7) Should we have a plot_plist field in AO? Then this can be combined
%       with the input plist.
%

function varargout = iplot(varargin)

  import utils.const.*

  %% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [upl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  %% Go through AOs and collect them into similar types

  tsAOs  = [];
```

```
fsAOs  = [];
xyAOs  = [];
xyzAOs = [];
cAOs   = [];

for j=1:numel(as)
  switch class(as(j).data)
    case 'tsdata'
      tsAOs = [tsAOs as(j)];
    case 'fsdata'
      fsAOs = [fsAOs as(j)];
    case 'xydata'
      xyAOs = [xyAOs as(j)];
    case 'xyzdata'
      xyzAOs = [xyzAOs as(j)];
    case 'cdata'
      cAOs = [cAOs as(j)];
    otherwise
      warning('!!! Unknown data type %s', class(as(j).data));
  end
end

%% Now plot all the objects on separate figures

hfig = [];
hax  = [];
hli  = [];

%----------- TSDATA
if ~isempty(tsAOs)
  % get default plist
  dpl = getDefaultPlist('Time-series plot');
  % combine the plists
  pl = combine(upl, dpl);
  % Call x-y plot
  [hf, ha, hl] = xy_plot(tsAOs, pl);
  hfig = [hfig hf];
  hax  = [hax ha];
  hli  = [hli hl];
end
%----------- XYDATA
if ~isempty(xyAOs)
  % get default plist
  dpl = getDefaultPlist('X-Y data plot');
  % combine the plists
  pl = combine(upl, dpl);
  % Call x-y plot
  [hf, ha, hl] = xy_plot(xyAOs, pl);
  hfig = [hfig hf];
  hax  = [hax ha];
  hli  = [hli hl];
end
%----------- XYZDATA
if ~isempty(xyzAOs)
  % get default plist
  dpl = getDefaultPlist('3D plot');
  % combine the plists
  pl = combine(upl, dpl);
  % Call x-y-z plot
  [hf, ha, hl] = xyz_plot(xyzAOs, pl);
  hfig = [hfig hf];
  hax  = [hax ha];
  hli  = [hli hl];
end
%----------- CDATA
if ~isempty(cAOs)
  % get default plist
  dpl = getDefaultPlist('Y data plot');
  % combine the plists
  pl = combine(upl, dpl);
  % Call x-y plot
  [hf, ha, hl] = y_plot(cAOs, pl);
  hfig = [hfig hf];
  hax  = [hax ha];
  hli  = [hli hl];
end
%----------- FSDATA
if ~isempty(fsAOs)
  % get default plist
  dpl = getDefaultPlist('Frequency-series plot');
  % combine the plists
  pl = combine(upl, dpl);
  % Call fsdata plot
  [hf, ha, hl] = fs_plot(fsAOs, pl);
  hfig = [hfig hf];
  hax  = [hax ha];
```

```
    hli  = [hli hl];
  end

  %% Deal with outputs
  if nargout == 1
    varargout{1} = hfig;
  end
  if nargout == 2
    varargout{1} = hfig;
    varargout{2} = hax;
  end
  if nargout == 3
    varargout{1} = hfig;
    varargout{2} = hax;
    varargout{3} = hli;
  end

  if nargout > 3
    error('### Incorrect number of outputs');
  end

end

%-------------------------------------------------------------------------
% Plot fsdata objects
%
function varargout = fs_plot(varargin)

  aos = varargin{1};
  pl  = varargin{2};

  % Extract parameters
  arrangement     = find(pl, 'Arrangement');
  colors          = find(pl, 'Colors');
  linecolors      = find(pl, 'LineColors');
  linestyles      = find(pl, 'LineStyles');
  markers         = find(pl, 'Markers');
  linewidths      = find(pl, 'LineWidths');
  legends         = find(pl, 'Legends');
  ylabels         = find(pl, 'YLabels');
  xlabels         = find(pl, 'XLabels');
  yscales         = find(pl, 'YScales');
  xscales         = find(pl, 'XScales');
  yranges         = find(pl, 'YRanges');
  xranges         = find(pl, 'XRanges');
  xmaths          = find(pl, 'XMaths');
  ymaths          =  find(pl, 'YMaths');
  type            = find(pl, 'Function');
  legendLoc       = find(pl, 'LegendLocation');
  complexPlotType = find(pl, 'complexPlotType');

  % check whether we want legends or not
  if iscell(legends)
    legendsOn = 1;
  else
    if strcmpi(legends, 'off')
      legendsOn = 0;
    else
      legendsOn = 1;
      legends = [];
    end
  end

  if ~iscell(linewidths), linewidths = {linewidths}; end
  if ~iscell(linestyles), linestyles = {linestyles}; end
  if ~iscell(linecolors), linecolors = {linecolors}; end
  if ~iscell(markers), markers = {markers}; end
  if ~iscell(legends), legends = {legends}; end
  if ~iscell(ylabels), ylabels = {ylabels}; end
  if ~iscell(xlabels), xlabels = {xlabels}; end
  if ~iscell(xmaths), xmaths = {xmaths}; end
  if ~iscell(ymaths), ymaths = {ymaths}; end
  if ~iscell(xscales), xscales = {xscales}; end
  if ~iscell(yscales), yscales = {yscales}; end
  if ~iscell(xranges), xranges = {xranges}; end
  if ~iscell(yranges), yranges = {yranges}; end

  % collect figure handles
  tsfig = []; tsax  = []; tsli  = [];
  % Legend holder
  legendStr = [];

  if ~isempty(aos)
    % Now loop over AOs
    Na = length(aos);
    % - first to check if any are complex
```

```
    haveComplex = 0;
    for j=1:Na
      y = aos(j).data.y;
      ymath = '';
      if j<=length(ymaths)
        if ~isempty(ymaths{j})
          eval(sprintf('y = %s;', ymaths{j}));
          ymath = ymaths{j};
        end
      end
      xmath = '';
      if j<=length(xmaths)
        if ~isempty(xmaths{j})
          eval(sprintf('x = %s;', xmaths{j}));
          xmath = xmaths{j};
        end
      end
      if ~isreal(y)
        haveComplex = 1;
      end
    end

    % Loop over the AOs now
    for j=1:Na
      % set real and imag subplot handles to empty
      tsax_r = [];
      tsax_i = [];
      %------- Apply math functions
      x = aos(j).data.x;
      y = aos(j).data.y;
      ymath = '';
      if j<=length(ymaths)
        if ~isempty(ymaths{j})
          eval(sprintf('y = %s;', ymaths{j}));
          ymath = ymaths{j};
        end
      end
      xmath = '';
      if j<=length(xmaths)
        if ~isempty(xmaths{j})
          eval(sprintf('x = %s;', xmaths{j}));
          xmath = xmaths{j};
        end
      end
      % what figures do we need?
      switch arrangement
        case 'single'
          tsfig = [tsfig figure];
          col = colors{1};
          % check if this data set is real or complex
          if ~isreal(y)
            % complex means we use two subplots
            tsax_r = subplot(2,1,1);
            tsax_i = subplot(2,1,2);
            tsax   = [tsax tsax_r tsax_i];
          else
            % real means we use a single subplot
            tsax_r = subplot(1, 1, 1);
            tsax = [tsax tsax_r];
          end
        case 'stacked'
          if j==1, tsfig = figure; end
          % if at least one of the input fsdata AOs is complex, we need to
          % allow for subplots
          if haveComplex
            tsax_r = subplot(2,1,1);
            tsax_i = subplot(2,1,2);
            tsax   = [tsax_r tsax_i];
          else
            tsax_r = subplot(1, 1, 1);
            tsax = tsax_r;
          end
          col = colors{mod(j-1,length(colors))+1};
          hold(tsax_r, 'on');
          if ishandle(tsax_i)
            hold(tsax_i, 'on');
          end
        case 'subplots'
          if j == 1, tsfig = figure; end
          c = 1+(j-1)*2;
          sx = Na;
          sy = 2;
          % Now we have one or two subplots per input object.
          if ~isreal(y)
            tsax_r = subplot(sx, sy,c);
            tsax_i = subplot(sx, sy,c+1);
```

```
            tsax   = [tsax tsax_r tsax_i];
          else
            tsax_r = subplot(sx, sy, c:c+1);
            tsax   = [tsax tsax_r];
          end
          col = colors{1};
        otherwise
          error('### Unknown plot arrangement');
      end

      %------- Plot the data

      % plot real or complex data and setup default values for scale and
      % labels as we go.
      if isreal(y)
        li   = feval(type, tsax_r, x, y);
        tsli = [tsli li];
        ylabelr = 'amplitude';
        ylabeli = 'imag';
        yscaleR = 'log';
        yscaleI = 'lin';
        xscaleR = 'log';
        xscaleI = 'lin';
      else
        switch complexPlotType
          case 'realimag'
            li = feval(type, tsax_r, x, y);
            li = [li feval(type, tsax_i, x, imag(y))];
            tsli = [tsli li];
            ylabelr = 'real';
            ylabeli = 'imag';
            yscaleR = 'lin';
            yscaleI = 'lin';
            xscaleR = 'lin';
            xscaleI = 'lin';
          case 'absdeg'
            li   = feval(type, tsax_r, x, abs(y));
            li   = [li feval(type, tsax_i, x, utils.math.phase(y))];
            tsli = [tsli li];
            ylabelr = 'amplitude';
            ylabeli = 'Phase [deg]';
            yscaleR = 'log';
            yscaleI = 'lin';
            xscaleR = 'log';
            xscaleI = 'log';
          case 'absrad'
            li   = feval(type, tsax_r, x, abs(y));
            li   = [li feval(type, tsax_i, x, angle(y))];
            tsli = [tsli li];
            ylabelr = 'amplitude';
            ylabeli = 'phase [rad]';
            yscaleR = 'log';
            yscaleI = 'lin';
            xscaleR = 'log';
            xscaleI = 'log';
          otherwise
            error('### Unknown plot type for complex data');
        end
      end

      %------- Axis properties

      % Set ylabel
      c = 1+(j-1)*2;
      if length(ylabels)==2 && strcmpi(ylabels{1}, 'all')
        ylstrR = ylabels{2};
      elseif c<=length(ylabels)
        if ~isempty(ylabels{c})
          ylstrR =  ylabels{c};
        else
          ylstrR = ylabelr;
        end
      else
        ylstrR = ylabelr;
      end
      if c<length(ylabels)
        if ~isempty(ylabels{c+1})
          ylstrI =  ylabels{c+1};
        else
          ylstrI = ylabeli;
        end
      else
        ylstrI = ylabeli;
      end
      if ~isempty(ymath)
        ymath = strrep(ymath, 'y', sprintf('%s', char(aos(j).data.yunits)));
```

```
        ylstrR = [ylstrR ' ' ymath ];
      else
        ylstrR = [ylstrR ' ' char(aos(j).data.yunits) ];
      end
      ylstrR = fixlabel(ylstrR);
      ylabel(tsax_r, ylstrR);
      if ishandle(tsax_i)
        ylstrI = fixlabel(ylstrI);
        ylabel(tsax_i, ylstrI);
      end

      % Set xlabel
      if length(xlabels)==2 && strcmpi(xlabels{1}, 'all')
        xlstr = xlabels{2};
      elseif j<=length(xlabels) && ~isempty(xlabels{j})
        xlstr =  xlabels{j};
      else
        xlstr = find(pl, 'XLabel');
      end
      if ~isempty(xmath)
        xmath = strrep(xmath, 'x', sprintf('%s', char(aos(j).data.xunits)));
        xlstr = [xlstr ' ' xmath ];
      else
        xlstr = [xlstr ' ' char(aos(j).data.xunits) ];
      end
      xlstr = fixlabel(xlstr);
      xlabel(tsax_r, xlstr);
      if ishandle(tsax_i)
        xlabel(tsax_i, xlstr);
      end

      % Set grid on or off
      grid(tsax_r, 'on');
      if ishandle(tsax_i)
        grid(tsax_i, 'on');
      end

      % Set Y scale
      if length(yscales)==2 && strcmpi(yscales{1}, 'all')
        yscaleR = yscales{2};
        yscaleI = yscales{2};
      else
        c = 1+(j-1)*2;
        if c<=length(yscales)
          if ~isempty(yscales{c})
            yscaleR =  yscales{c};
          end
        end
        if c<length(yscales)
          if ~isempty(yscales{c+1})
            yscaleI =  yscales{c+1};
          end
        end
      end

      set(tsax_r, 'YScale', yscaleR);
      if ishandle(tsax_i)
        set(tsax_i, 'YScale', yscaleI);
      end

      % Set X scale
      if length(xscales)==2 && strcmpi(xscales{1}, 'all')
        xscaleR = xscales{2};
        xscaleI = xscales{2};
      else
        c = 1+(j-1)*2;
        if c<=length(xscales)
          if ~isempty(xscales{c})
            xscaleR =  xscales{c};
          end
        end
        if c<length(xscales)
          if ~isempty(xscales{c+1})
            xscaleI =  xscales{c+1};
          end
        end
      end
      set(tsax_r, 'XScale', xscaleR);
      if ishandle(tsax_i)
        set(tsax_i, 'XScale', xscaleI);
      end

      % Set Y range
      if length(yranges)==2 && strcmpi(yranges{1}, 'all')
        set(tsax_r, 'YLim', yranges{2});
        if ishandle(tsax_i)
```

```
            set(tsax_i, 'YLim', yranges{2});
          end
        else
          c = 1+(j-1)*2;
          if c<=length(yranges)
            if ~isempty(yranges{c})
              set(tsax_r, 'YLim', yranges{c});
            end
          end
          if c<length(yranges)
            if ~isempty(yranges{c+1})
              if ishandle(tsax_i)
                set(tsax_i, 'YLim', yranges{c+1});
              end
            end
          end
        end

        % Set X range
        if length(xranges)==2 && strcmpi(xranges{1}, 'all')
          set(tsax_r, 'XLim', xranges{2});
          if ishandle(tsax_i)
            set(tsax_i, 'XLim', xranges{2});
          end
        else
          c = 1+(j-1)*2;
          if c<=length(xranges)
            if ~isempty(xranges{c})
              set(tsax_r, 'XLim', xranges{c});
            end
          end
          if c<length(xranges)
            if ~isempty(xranges{c+1})
              if ishandle(tsax_i)
                set(tsax_i, 'XLim', xranges{c+1});
              end
            end
          end
        end

        %------- line properties

        % Set line color
        if isreal(y)
          set(li, 'Color', col);
        else
          set(li, 'Color', col);
          set(li, 'Color', col);
        end

        % Overide with user colors
        if length(linecolors) == 2 && strcmpi(linecolors{1}, 'all')
          set(li, 'Color', linecolors{2});
        else
          if j<=length(linecolors) && ~isempty(linecolors{j})
            if isreal(y)
              set(li, 'Color', linecolors{j});
            else
              set(li, 'Color', linecolors{j});
              set(li, 'Color', linecolors{j});
            end
          end
        end

        % Set line style
        if length(linestyles) == 2 && strcmpi(linestyles{1}, 'all')
          set(li, 'LineStyle', linestyles{2});
        else
          if j<=length(linestyles) && ~isempty(linestyles{j})
            if isreal(y)
              set(li, 'LineStyle', linestyles{j});
            else
              set(li, 'LineStyle', linestyles{j});
              set(li, 'LineStyle', linestyles{j});
            end
          end
        end

        % Set line widths
        if length(linewidths) == 2 && strcmpi(linewidths{1}, 'all')
          set(li, 'LineWidth', linewidths{2});
        else
          if j<=length(linewidths) && ~isempty(linewidths{j})
            if isreal(y)
              set(li, 'LineWidth', linewidths{j});
            else
```

```
            set(li, 'LineWidth', linewidths{j});
            set(li, 'LineWidth', linewidths{j});
          end
        end
      end

      % Set markers
      if length(markers) == 2 && strcmpi(markers{1}, 'all')
        set(li, 'Marker', markers{2});
      else
        if j<=length(markers) && ~isempty(markers{j})
          if isreal(y)
            set(li, 'Marker', markers{j});
          else
            set(li, 'Marker', markers{j});
            set(li, 'Marker', markers{j});
          end
        end
      end

      % Set legend string
      lstr = '';
      if legendsOn
        if j<=length(legends) && ~isempty(legends{j})
          lstr = legends{j};
        else
          lstr = utils.plottools.label(aos(j).name);
        end
      end
      legendStr = [legendStr cellstr(lstr)];

      % Set the legend now if we can
      if legendsOn
        if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
          legend(tsax_r, fixlabel(legendStr{end}), 'Location', legendLoc);
        end
      end
    end % End loop over AOs

    % Process legends for stacked plots
    if legendsOn
      if strcmp(arrangement, 'stacked')
        h = legend(tsax_r, fixlabel(legendStr), 'Location', legendLoc);
        set(h, 'FontSize', 10)
      end
    end
  end % End empty AOs

  % Set outputs
  if nargout > 0
    varargout{1} = tsfig;
  end
  if nargout > 1
    varargout{2} = tsax;
  end
  if nargout == 3
    varargout{3} = tsli;
  end
  if nargout > 3
    error('### Too many output arguments');
  end
end % End fs_plot

%-------------------------------------------------------------------------
% Plot tsdata and xydata objects
%
function varargout = xy_plot(varargin)

  aos = varargin{1};
  pl  = varargin{2};

  % Extract parameters
  arrangement = find(pl, 'Arrangement');
  linecolors  = find(pl, 'LineColors');
  colors      = find(pl, 'Colors');
  linestyles  = find(pl, 'LineStyles');
  markers     = find(pl, 'Markers');
  linewidths  = find(pl, 'Line1Widths');
  legends     = find(pl, 'Legends');
  ylabels     = find(pl, 'YLabels');
  xlabels     = find(pl, 'XLabels');
  xmaths      = find(pl, 'XMaths');
  ymaths      = find(pl, 'YMaths');
  yranges     = find(pl, 'YRanges');
  xranges     = find(pl, 'XRanges');
  yscales     = find(pl, 'YScales');
```

```
xscales     = find(pl, 'XScales');
type        = find(pl, 'Function');
legendLoc   = find(pl, 'LegendLocation');

% check whether we want legends or not
if iscell(legends)
  legendsOn = 1;
else
  if strcmpi(legends, 'off')
    legendsOn = 0;
  else
    legendsOn = 1;
    legends = [];
  end
end

if ~iscell(linewidths), linewidths = {linewidths}; end
if ~iscell(linestyles), linestyles = {linestyles}; end
if ~iscell(linecolors), linecolors = {linecolors}; end
if ~iscell(markers), markers = {markers}; end
if ~iscell(legends), legends = {legends}; end
if ~iscell(ylabels), ylabels = {ylabels}; end
if ~iscell(xlabels), xlabels = {xlabels}; end
if ~iscell(xmaths), xmaths = {xmaths}; end
if ~iscell(ymaths), ymaths = {ymaths}; end
if ~iscell(xranges), xranges = {xranges}; end
if ~iscell(yranges), yranges = {yranges}; end
if ~iscell(xscales), xscales = {xscales}; end
if ~iscell(yscales), yscales = {yscales}; end

% collect figure handles
tsfig = []; tsax  = []; tsli  = [];
% Legend holder
legendStr = [];
if ~isempty(aos)
  % Now loop over AOs to get earliest start time
  if strcmp(arrangement, 'stacked')
    T0 = 1e50;
    Na = length(aos);
    for jj=1:Na
      % Get this AO
      if isa(aos(jj).data, 'tsdata')
        if aos(jj).data.t0.utc_epoch_milli/1000 < T0
          T0 = floor(aos(jj).data.t0.utc_epoch_milli/1000);
        end
      end
    end
  else
    T0 = 0;
  end

  % Now loop over AOs
  Na = length(aos);
  for j=1:Na
    % Get this AO
    toff = 0;
    % what figures do we need?
    switch arrangement
      case 'single'
        tsfig = [tsfig figure];
        tsax = subplot(1,1,1);
        col = colors{1};
        if isa(aos(j).data, 'tsdata')
          torigin = aos(j).data.t0;
        end
      case 'stacked'
        if j==1, tsfig = figure; end
        tsax = subplot(1,1,1);
        col = colors{mod(j-1,length(colors))+1};
        hold on;
        % deal with time-stamps here
        if isa(aos(j).data, 'tsdata')
          toff = aos(j).data.t0.utc_epoch_milli/1000 - T0;
        else
          toff = 0;
        end
        if isa(aos(j).data, 'tsdata')
          torigin = time(T0*1000);
        end
      case 'subplots'
        if j == 1, tsfig = figure; end
        tsax = [tsax subplot(Na, 1, j)];
        col = colors{1};
        if isa(aos(j).data, 'tsdata')
          torigin = aos(j).data.t0;
        end
```

```
          otherwise
            error('### Unknown plot arrangement');
        end

        %------- Apply math functions

        % need t0 offset for this time-series
        x = aos(j).data.getX + toff;
        y = aos(j).data.y;

        ymath = '';
        if j<=length(ymaths)
          if ~isempty(ymaths{j})
            eval(sprintf('y = %s;', ymaths{j}));
            ymath = ymaths{j};
          end
        end
        xmath = '';
        if j<=length(xmaths)
          if ~isempty(xmaths{j})
            eval(sprintf('x = %s;', xmaths{j}));
            xmath = xmaths{j};
          end
        end

        %------- Plot the data

        li   = feval(type, tsax(end), x, y);
        tsli = [tsli li];
        if isa(aos(j).data, 'tsdata')
          title(sprintf('Time origin: %s', char(torigin)));
        end
        %------- Axis properties

        % Set ylabel
        if j<=length(ylabels) && ~isempty(ylabels{j})
          ylstr =  ylabels{j};
        else
          ylstr = find(pl, 'YLabel');
        end
        if ~isempty(ymath)
          ymath = strrep(ymath, 'y', sprintf('%s', char(aos(j).data.yunits)));
          ylstr = [ylstr ' ' ymath ];
        else
          ylstr = [ylstr ' ' char(aos(j).data.yunits)];
        end
        ylstr = fixlabel(ylstr);
        ylabel(ylstr);

        % Set xlabel
        if j<=length(xlabels) && ~isempty(xlabels{j})
          xlstr =  xlabels{j};
        else
          xlstr = find(pl, 'XLabel');
        end
        if ~isempty(xmath)
          xmath = strrep(xmath, 'x', sprintf('%s', char(aos(j).data.xunits)));
          xlstr = [xlstr ' ' xmath  ];
        else
          xlstr = [xlstr ' ' char(aos(j).data.xunits)  ];
        end
        xlstr = fixlabel(xlstr);
        xlabel(xlstr);

        % Set Y range
        if length(yranges) == 2 && strcmpi(yranges{1}, 'all')
          set(tsax(end), 'YLim', yranges{2});
        else
          if j<=length(yranges)
            if ~isempty(yranges{j})
              set(tsax(end), 'YLim', yranges{j});
            end
          end
        end

        % Set Y scale
        yscaleR = 'lin';
        if length(yscales)==2 && strcmpi(yscales{1}, 'all')
          yscaleR = yscales{2};
        else
          if j<=length(yscales)
            if ~isempty(yscales{j})
              yscaleR =  yscales{j};
            end
          end
        end
```

```
      set(tsax(end), 'YScale', yscaleR);

      % Set X scale
      xscaleR = 'lin';
      if length(xscales)==2 && strcmpi(xscales{1}, 'all')
        xscaleR = xscales{2};
      else
        if j<=length(xscales)
          if ~isempty(xscales{j})
            xscaleR =  xscales{j};
          end
        end
      end
      set(tsax(end), 'XScale', xscaleR);

      % Set X range
      if length(xranges) == 2 && strcmpi(xranges{1}, 'all')
        set(tsax(end), 'XLim', xranges{2});
      else
        if j<=length(xranges)
          if ~isempty(xranges{j})
            set(tsax(end), 'XLim', xranges{j});
          end
        end
      end

      % Set grid on or off
      grid(tsax(end), 'on');

      %------- line properties

      % Set line color
      set(li, 'Color', col);

      % Set specific colors
      if length(linecolors) == 2 && strcmpi(linecolors{1}, 'all')
        set(li, 'Color', linecolors{2});
      else
        if j<=length(linecolors) && ~isempty(linecolors{j})
          set(li, 'Color', linecolors{j});
        end
      end

      % Set line style
      if length(linestyles) == 2 && strcmpi(linestyles{1}, 'all')
        set(li, 'LineStyle', linestyles{2});
      else
        if j<=length(linestyles) && ~isempty(linestyles{j})
          set(li, 'LineStyle', linestyles{j});
        end
      end

      % Set markers
      if length(markers) == 2 && strcmpi(markers{1}, 'all')
        set(li, 'Marker', markers{2});
      else
        if j<=length(markers) && ~isempty(markers{j})
          set(li, 'Marker', markers{j});
        end
      end

      % Set line widths
      if length(linewidths) == 2 && strcmpi(linewidths{1}, 'all')
        set(li, 'LineWidth', linewidths{2});
      else
        if j<=length(linewidths) && ~isempty(linewidths{j})
          set(li, 'LineWidth', linewidths{j});
        end
      end

      % Set legend string
      lstr = '';
      if legendsOn
        if j<=length(legends) && ~isempty(legends{j})
          lstr = legends{j};
        else
          lstr = utils.plottools.label(aos(j).name);
        end
      end
      legendStr = [legendStr cellstr(lstr)];

      % Set the legend now if we can
      if legendsOn
        if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
          legend(fixlabel(legendStr{end}), 'Location', legendLoc);
        end
```

```
      end
    end

    % Process legends for stacked plots
    if legendsOn
      if strcmp(arrangement, 'stacked')
        h = legend(fixlabel(legendStr), 'Location', legendLoc);
        set(h, 'FontSize', 10)
      end
    end
  end % End if empty AOs

  % Set outputs
  if nargout > 0
    varargout{1} = tsfig;
  end
  if nargout > 1
    varargout{2} = tsax;
  end
  if nargout == 3
    varargout{3} = tsli;
  end
  if nargout > 3
    error('### Too many output arguments');
  end
end % end xy_plot


%-------------------------------------------------------------------------
% Plot cdata objects
%
function varargout = y_plot(varargin)

  aos = varargin{1};
  pl  = varargin{2};

  % Extract parameters
  arrangement = find(pl, 'Arrangement');
  linecolors  = find(pl, 'LineColors');
  colors      = find(pl, 'Colors');
  linestyles  = find(pl, 'LineStyles');
  markers     = find(pl, 'Markers');
  linewidths  = find(pl, 'LineWidths');
  legends     = find(pl, 'Legends');
  ylabels     = find(pl, 'YLabels');
  xlabels     = find(pl, 'XLabels');
  xmaths      = find(pl, 'XMaths');
  ymaths      = find(pl, 'YMaths');
  yranges     = find(pl, 'YRanges');
  xranges     = find(pl, 'XRanges');
  yscales     = find(pl, 'YScales');
  xscales     = find(pl, 'XScales');
  type        = find(pl, 'Function');
  legendLoc   = find(pl, 'LegendLocation');

  if ~iscell(xscales), xscales = {xscales}; end
  if ~iscell(yscales), yscales = {yscales}; end

  % check whether we want legends or not
  if iscell(legends)
    legendsOn = 1;
  else
    if strcmp(legends, 'off')
      legendsOn = 0;
    else
      legendsOn = 1;
      legends = [];
    end
  end

  if ~iscell(linewidths), linewidths = {linewidths}; end
  if ~iscell(linestyles), linestyles = {linestyles}; end
  if ~iscell(linecolors), linecolors = {linecolors}; end
  if ~iscell(markers), markers = {markers}; end
  if ~iscell(legends), legends = {legends}; end
  if ~iscell(ylabels), ylabels = {ylabels}; end
  if ~iscell(xlabels), xlabels = {xlabels}; end
  if ~iscell(xmaths), xmaths = {xmaths}; end
  if ~iscell(ymaths), ymaths = {ymaths}; end
  if ~iscell(xranges), xranges = {xranges}; end
  if ~iscell(yranges), yranges = {yranges}; end

  % collect figure handles
  tsfig = []; tsax  = []; tsli  = [];
  % Legend holder
  legendStr = [];
```

```
  if ~isempty(aos)
    % Now loop over AOs
    Na = length(aos);
    for j=1:Na
      % what figures do we need?
      switch arrangement
        case 'single'
          tsfig = [tsfig figure];
          tsax = subplot(1,1,1);
          col = colors{1};
        case 'stacked'
          if j==1, tsfig = figure; end
          tsax = subplot(1,1,1);
          col = colors{mod(j-1,length(colors))+1};
          hold on;
        case 'subplots'
          if j == 1, tsfig = figure; end
          tsax = [tsax subplot(Na, 1, j)];
          col = colors{1};
        otherwise
          error('### Unknown plot arrangement');
      end

      %------- Apply math functions
      if isreal(aos(j).data.y)
        x = 1:length(aos(j).data.y);
        y = aos(j).data.y;
      else
        x = real(aos(j).data.y);
        y = imag(aos(j).data.y);
      end

      ymath = '';
      if j<=length(ymaths)
        if ~isempty(ymaths{j})
          eval(sprintf('y = %s;', ymaths{j}));
          ymath = ymaths{j};
        end
      end
      xmath = '';
      if j<=length(xmaths)
        if ~isempty(xmaths{j})
          eval(sprintf('x = %s;', xmaths{j}));
          xmath = xmaths{j};
        end
      end

      %------- Plot the data

      idcs = feval(type, tsax(end), x, y);
      tsli = [tsli idcs(1:end).'];

      %------- Axis properties

      % Set ylabel
      if j<=length(ylabels) && ~isempty(ylabels{j})
        ylstr =  ylabels{j};
      else
        ylstr = find(pl, 'YLabel');
      end
      if ~isempty(ymath)
        ymath = strrep(ymath, 'y', sprintf('%s', char(aos(j).data.yunits)));
        ylstr = [ylstr ' ' ymath ];
      else
        ylstr = [ylstr ' ' char(aos(j).data.yunits) ];
      end
      ylstr = fixlabel(ylstr);
      ylabel(ylstr);

      % Set xlabel
      if j<=length(xlabels) && ~isempty(xlabels{j})
        xlstr =  xlabels{j};
      else
        xlstr = find(pl, 'XLabel');
      end
      if ~isempty(xmath)
        xmath = strrep(xmath, 'x', 'N');
        xlstr = [xlstr ' ' xmath ];
      else
        xlstr = [xlstr ' [N]' ];
      end
      xlstr = fixlabel(xlstr);
      xlabel(xlstr);

      % Set Y scale
      yscaleR = 'lin';
```

```
      if length(yscales)==2 && strcmpi(yscales{1}, 'all')
        yscaleR = yscales{2};
      else
        if j<=length(yscales)
          if ~isempty(yscales{j})
            yscaleR =  yscales{j};
          end
        end
      end
      set(tsax(end), 'YScale', yscaleR);

      % Set X scale
      xscaleR = 'lin';
      if length(xscales)==2 && strcmpi(xscales{1}, 'all')
        xscaleR = xscales{2};
      else
        if j<=length(xscales)
          if ~isempty(xscales{j})
            xscaleR =  xscales{j};
          end
        end
      end
      set(tsax(end), 'XScale', xscaleR);

      % Set Y range
      if length(yranges) == 2 && strcmpi(yranges{1}, 'all')
        set(tsax(end), 'YLim', yranges{2});
      else
        if j<=length(yranges)
          if ~isempty(yranges{j})
            set(tsax(end), 'YLim', yranges{j});
          end
        end
      end

      % Set X range
      if length(xranges) == 2 && strcmpi(xranges{1}, 'all')
        set(tsax(end), 'XLim', xranges{2});
      else
        if j<=length(xranges)
          if ~isempty(xranges{j})
            set(tsax(end), 'XLim', xranges{j});
          end
        end
      end

      % Set grid on or off
      grid(tsax(end), 'on');

      %------- line properties

      % Set line color
      set(idcs, 'Color', col);

      % Overide line colors with user defined colors
      if length(linecolors) == 2 && strcmpi(linecolors{1}, 'all')
        set(idcs, 'Color', linecolors{2});
      else
        if j<=length(linecolors) && ~isempty(linecolors{j})
          set(idcs, 'Color', linecolors{j});
        end
      end

      % Set line style
      if length(linestyles) == 2 && strcmpi(linestyles{1}, 'all')
        set(idcs, 'LineStyle', linestyles{2});
      else
        if j<=length(linestyles) && ~isempty(linestyles{j})
          set(idcs, 'LineStyle', linestyles{j});
        end
      end

      % Set Markers
      if length(markers) == 2 && strcmpi(markers{1}, 'all')
        set(idcs, 'Marker', markers{2});
      else
        if j<=length(markers) && ~isempty(markers{j})
          set(idcs, 'Marker', markers{j});
        end
      end

      % Set line widths
      if length(linewidths) == 2 && strcmpi(linewidths{1}, 'all')
        set(idcs, 'LineWidth', linewidths{2});
      else
        if j<=length(linewidths) && ~isempty(linewidths{j})
```

```
          set(idcs, 'LineWidth', linewidths{j});
        end
      end

      % Set legend string
      if legendsOn
        if j<=length(legends) && ~isempty(legends{j})
          legendStr = [legendStr legends(j)];
        else
          lstr = utils.plottools.label(aos(j).name);
          legendStr = [legendStr cellstr(lstr)];
        end
      end

      % Set the legend now if we can
      if legendsOn
        if strcmp(arrangement, 'single') || strcmp(arrangement, 'subplots')
          legend(fixlabel(legendStr{end}), 'Location', legendLoc);
        end
      end
    end

    % Process legends for stacked plots
    if legendsOn
      if strcmp(arrangement, 'stacked')
        h = legend(fixlabel(legendStr), 'Location', legendLoc);
        set(h, 'FontSize', 10)
      end
    end

  end

  % Set outputs
  if nargout > 0
    varargout{1} = tsfig;
  end
  if nargout > 1
    varargout{2} = tsax;
  end
  if nargout == 3
    varargout{3} = tsli;
  end
  if nargout > 3
    error('### Too many output arguments');
  end
end % End y_plot

%-----------------------------------------------------------------------
% Plot xyzdata objects
%
function varargout = xyz_plot(varargin)

  aos = varargin{1};
  pl  = varargin{2};

  % Extract parameters
  arrangement = find(pl, 'Arrangement');
  linecolors  = find(pl, 'LineColors');
  colors      = find(pl, 'Colors');
  linestyles  = find(pl, 'LineStyles');
  linewidths  = find(pl, 'LineWidths');
  legends     = find(pl, 'Legends');
  zlabels     = find(pl, 'ZLabels');
  ylabels     = find(pl, 'YLabels');
  xlabels     = find(pl, 'XLabels');
  xmaths      = find(pl, 'XMaths');
  ymaths      = find(pl, 'YMaths');
  zmaths      = find(pl, 'ZMaths');
  legendLoc   = find(pl, 'LegendLocation');

  % check whether we want legends or not
  if iscell(legends)
    legendsOn = 1;
  else
    if strcmp(legends, 'off')
      legendsOn = 0;
    else
      legendsOn = 1;
      legends = [];
    end
  end

  % collect figure handles
  tsfig = [];
  tsax  = [];
  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
```

```
        xlstr = [xlstr ' ' char(aos(j).data.xunits) ];
      end
      xlstr = fixlabel(xlstr);
      xlabel(xlstr);

      % Set grid on or off
      grid(tsax(end), 'on');

      % Set title string
      if ~strcmpi(legends, 'off')
        if j<=length(legends) && ~isempty(legends{j})
          legendStr = [legendStr legends(j)];
        else
          lstr = utils.plottools.label(aos(j).name);
          legendStr = [legendStr cellstr(lstr)];
        end
      end

      % Set the legend now if we can
      tstr = legendStr{end};
      if legendsOn
        title(tstr);
      end

      % Set colorbars
      hc = colorbar('peer', tsax(end));
      if j<=length(zlabels)
        if ~isempty(zlabels{j})
          zlstr = zlabels{j};
        end
      else
        zlstr = find(pl, 'Zlabel');
      end
      if ~isempty(zmath), zlstr = [zlstr sprintf('\n%s', zmath)]; end
      ylh = get(hc, 'YLabel');
      set(ylh, 'String', zlstr);
      set(ylh, 'Fontsize', get(tsax(end), 'Fontsize'))
      set(ylh, 'FontName', get(tsax(end), 'FontName'))
      set(ylh, 'FontAngle', get(tsax(end), 'FontAngle'))
      set(ylh, 'FontWeight', get(tsax(end), 'FontWeight'))
    end
  end

  % Set outputs
  if nargout > 0
    varargout{1} = tsfig;
  end
  if nargout > 1
    varargout{2} = tsax;
  end
  if nargout == 3
    varargout{3} = tsli;
  end
  if nargout > 3
    error('### Too many output arguments');
  end
end % end xyz_plot

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  elseif nargin == 1&& ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pl = getDefaultPlist(sets{1});
  else
    sets = {'Time-series plot', 'Frequency-series plot', 'Y data plot', 'X-Y data plot', '3D plot'};
    % get plists
    pl(size(sets)) = plist;
    for k = 1:numel(sets)
      pl(k) =  getDefaultPlist(sets{k});
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.output, '$Id: iplot.m,v 1.48 2008/09/05 14:14:33 hewitson Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function out = getDefaultPlist(set)
```

```
  % Get the LTPDA color set for lines
  colors = getappdata(0,'ltpda_default_plot_colors');

  out = plist(...
    'Colors', colors, ...
    'Arrangement', 'stacked', ...
    'Function', 'plot', 'LegendLocation', 'Best');

  switch set
    case 'Frequency-series plot'
      out.append(...
        'complexPlotType', 'absdeg', ...
        'XLabel', 'Frequency');
    case 'Time-series plot'
      out.append(...
        'Xlabel', 'Time', ...
        'Ylabel', 'Amplitude');
    case 'X-Y data plot'
      out.append(...
        'Xlabel', 'X-data', ...
        'Ylabel', 'Y-data', ...
        'YMaths', '', ...
        'XMaths', '');
    case '3D plot'
      out = plist(...
        'Colors', colors,...
        'Arrangement', 'single', ...
        'Xlabel', 'Time', ...
        'Ylabel', 'Frequency',...
        'Zlabel', 'Amplitude', ...
        'YMaths', '', ...
        'ZMaths', '', ...
        'XMaths', '');
    case 'Y data plot'
      out.append(...
        'Xlabel', 'Index', ...
        'Ylabel', 'Value');
    otherwise
      out = plist();
  end
end

% Perform some substitutions on the labels
function ss = fixlabel(ss)

  wasCell = true;
  if ~iscell(ss)
    ss = {ss};
    wasCell = false;
  end

  for kk=1:numel(ss)
    s = ss{kk};

    % Replace all ^(...) with ^{...}
    j = 1;
    while j<numel(s)
      if strcmp(s(j:j+1), '^(')
        % find next )
        for k=1:numel(s)-j+1
          if s(j+k) == ')'
            s(j+1) = '{';
            s(j+k) = '}';
            break;
          end
        end
      end
      j = j + 1;
    end
    % Replace all .^ with ^
    s = strrep(s, '.^', '^');

    ss(kk) = {s};
  end


  if ~wasCell
    ss = ss{1};
  end

end
```

plot

PURPOSE ^

PLOT a simple plot of analysis objects.

SYNOPSIS ^

function varargout = plot(varargin)

DESCRIPTION ^

```
 PLOT a simple plot of analysis objects.
%%%
```

Description of plot

```
0001 % PLOT a simple plot of analysis objects.
0002 %%%%
```

```
    end

    %%%%%%%%%%   Get the x-, y-values   %%%%%%%%%%
    x = ao_s(ii).data.getX;
    y = ao_s(ii).data.y;

    if isreal(y)
      if isempty(x)
        x = 1:length(y);
      end
    else % iscomplex(y)
      if isempty(x)
        x = real(y);
        y = imag(y);
      end
    end

    %%%%%%%%%%   Call the different plot command lines   %%%%%%%%%%
    if ~isempty(axes_handle) && ~isempty(line_spec)
      line_h = plot(axes_handle, x, y, line_spec);
    elseif ~isempty(axes_handle)
      line_h = plot(axes_handle, x, y);
    elseif ~isempty(line_spec)
      line_h = plot(x, y, line_spec);
    else
      line_h = plot(x, y);
    end

    hold on;

    line_h  = reshape(line_h, 1, []);
    lines_h = [lines_h line_h];

    %%%%%%%%%%   Set the different handles: axes- and figure- handle   %%%%%%%%%%
    axes_h   = get(line_h(1), 'Parent');
    figure_h = get(axes_h,    'Parent');

    %%%%%%%%%%   Set line series Properties OR axes Properties   %%%%%%%%%%
    while length(prop_val) >= 2
      prop = prop_val{1};
      val  = prop_val{2};
      prop_val = prop_val(3:end);
      if isprop(line_h, prop)
        set(line_h, prop, val);
      else
        if isprop(axes_h, prop)
          set(axes_h, prop, val);
        else
          error('### Invalid line property OR axes property: ''%s''.', prop);
        end
      end
    end

  end % for ii = 1:length(ao_s)


  %%%%%%%%%%   varargout{1} = line   handle   %%%%%%%%%%
  %%%%%%%%%%   varargout{2} = axes   handle   %%%%%%%%%%
  %%%%%%%%%%   varargout{3} = figure handle   %%%%%%%%%%
  if nargout == 0
  elseif nargout == 1
    varargout{1} = lines_h;
  elseif nargout == 2
    varargout{1} = lines_h;
    varargout{2} = axes_h;
  elseif nargout == 3
    varargout{1} = lines_h;
    varargout{2} = axes_h;
    varargout{3} = figure_h;
  else
    error('### Unknown number of outputs.')
  end

  hold off;

end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.output, '$Id: plot.m,v 1.36 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist();
end

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

save

PURPOSE ^

SAVE overloads save operator for ltpda objects.

SYNOPSIS ^

function varargout = save(varargin)

DESCRIPTION ^

```
 SAVE overloads save operator for ltpda objects.
%%%
```

```
%
% 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
%
%%%
```

```
      utils.helper.xmlwrite(objs, xml, parent, '');    % Save the XML document.
      xmlwrite(filename, xml);

    otherwise
      error('### unknown file extension.');
  end

end

%%%
```

type

PURPOSE ^

TYPE converts the input objects to MATLAB functions.

SYNOPSIS ^

function varargout = type(varargin)

DESCRIPTION ^

```
 TYPE converts the input objects to MATLAB functions.
%%%
```

```
% 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
%
%%%
```

```
      end

      % Footer
      disp(' ');
      disp(utils.prog.strpad('', [LL LL], '-'));
      disp(utils.prog.strpad(as(j).name, [LL LL], '-'));
      disp(utils.prog.strpad('', [LL LL], '-'));
      disp(' ');
    end

  %----- Output to file
  else

    % clear this m-file from memory
    clear(filename);

    % Open the file
    ii = getInfo;
    fd = fopen(filename, 'w+');
    [path,name,ext,vers] = fileparts(filename);

    % Write help header
    fprintf(fd, '%% %s \n', upper(filename));
    fprintf(fd, '%% \n');
    fprintf(fd, '%% \n');
    fprintf(fd, '%% written by %s / %s\n', mfilename, ii.mversion);
    fprintf(fd, '%% \n');
    fprintf(fd, '%% \n');
    fprintf(fd, ' \n');
    fprintf(fd, ' \n');

    % write main fcn
    fprintf(fd, 'function out = %s\n\n', name);
    fprintf(fd, 'out = [];\n');
    for j=1:numel(as)
      fprintf(fd, 'out = [out obj%03d];\n', j);
    end
    fprintf(fd, '\nend\n\n');

    % Write each sub function
    for kk=1:numel(fcn)
      % header
      fprintf(fd, '%% sub-function for object %s\n', as(kk).name);
      fprintf(fd, 'function a_out = obj%03d\n\n', kk);
      % write command for this function
      for jj=numel(fcn(kk).cmds):-1:1
        fprintf(fd, '\t%s\n', fcn(kk).cmds{jj});
      end
      % Footer
      fprintf(fd, '\nend\n\n');
    end

    fprintf(fd, '\n\n%% END\n');

    % Close file
    fclose(fd);
    % This somehow forces the new file to be available to MATLAB
    cd ('.');
  end

end

%-----------------------------------------------------------------------------
% Get Info Object
%-----------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.output, '$Id: type.m,v 1.5
2008/09/04 15:29:30 ingo Exp $', sets, pl);
end

%-----------------------------------------------------------------------------
% Get Default Plist
%-----------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('filename', '');
end
```

```
0177
0178 % END
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%
```

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.
%%%
```

```
% 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
%
%%%
```

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 ^

```
% 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
%
%%%
```

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.
%%%
```

```
% 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
%
%%%
```

eq

PURPOSE ^

EQ overloads the == operator for ltpda objects.

SYNOPSIS ^

function result = eq(obj1, obj2, varargin)

DESCRIPTION ^

```
 EQ overloads the == operator for ltpda objects.
%%%
```

SOURCE CODE ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function result = eq(obj1, obj2, varargin)

  % Check if this is a call for parameters
  hh = {obj1, obj2, varargin{:}};
  if utils.helper.isinfocall(hh{:})
    result = getInfo(varargin{1});
    return
  end

  import utils.const.*

  %%%%% Check class
  if ~strcmp(class(obj1), class(obj2))
    utils.helper.msg(msg.PROC1, 'NOT EQUAL: The objects are not from the same class. [%s] <-> [%s]', class(obj1), class(obj2));
    result = 0;
    return
  end

  %%%%% Check length of obj1 and obj2
  if numel(obj1) ~= numel(obj2)
    utils.helper.msg(msg.PROC1, 'NOT EQUAL: The size of the %s-object''s. [%d] <-> [%d]', class(obj1), numel(obj1), numel(obj2));
    result = 0;
    return
  end

  exception_list = varargin;

  %%%%% Convert a potential existing plist into a exception
  if ~isempty(exception_list) && isa(exception_list{1}, 'plist')
    exception_list = find(exception_list{1}, 'Exceptions');
    if isempty(exception_list)
      exception_list = cell(0);
    elseif ~iscell(exception_list)
      exception_list = cellstr(exception_list);
    end
  end

  %%%%% for each element in obj1 and obj2
  for jj = 1:numel(obj1)

    result = 1;

    fields = fieldnames(obj1(jj));

    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
0149        end
0150      end
0151
0152    end
0153  end
0154
0155  %%%%%%%%%%
```

```
% DESCRIPTION: Get Info Object
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%
```

ne

PURPOSE ^

NE overloads the ~= operator for ltpda objects.

SYNOPSIS ^

function result = ne(obj1, obj2, varargin)

DESCRIPTION ^

```
 NE overloads the ~= operator for ltpda objects.
%%%
```

SOURCE CODE ^

```
% NE overloads the ~= operator for ltpda objects.
%%%
```

```
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%
```

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
%%%
```

```
% COHERE makes coherence estimates of the time-series objects
%%%
```

```
  ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: cohere.m,v 1.10 2008/09/05 11:05:28 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist(...
    'Nfft', -1, ...
    'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
    'Olap', -1, ...
    'Order', 0);
end
% END

```

*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.
%%%
```

function ii = getInfo(varargin)
function pl = getDefaultPlist()

SOURCE CODE ^

```
% COMPUTE performs the given operations on the input AOs.
%%%
```

```
    ops = find(pl, 'Operations');
  end

  % Check we have a cell array
  if ischar(ops)
    ops = {ops};
  end

  % Loop over operations
  bs(length(ops),1) = ao;
  for jj=1:length(ops)
    % evaluate this operation
    if ops{jj}(end) ~= ';'
      ops{jj}(end+1) = ';';
    end
    bs(jj) = eval(ops{jj});
  end

  % Set outputs
  varargout{1} = bs;
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: compute.m,v 1.11 2008/09/05 11:05:28 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('Operations', 'a');
end
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
- interpmissing 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 ^

```
% CONSOLIDATE resamples all input AOs onto the same time grid.
%%%
```

```
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
0130    %----------------- Resample all vectors on to the same grid
0131    utils.helper.msg(msg.PROC1, 'resample to same grid');
0132    % compute new time grid
0133    for j=1:na
0134      N = length(bs(j).data.getX);
0135      t = linspace(0, (N-1)/fs, N);
0136      interp(bs(j), plist('vertices', t));
0137    end
0138
0139    %---------------- Time properties of AOs
0140    % Find max start time
0141    start = 0;
0142    for j=1:na
0143      dstart = bs(j).data.t0.utc_epoch_milli/1000 + bs(j).data.getX(1);
0144      if dstart > start
0145        start = dstart;
0146      end
0147    end
0148
0149    % Find min stop time
0150    stop = 1e20;
0151    for j=1:na
0152      dstop = floor(bs(j).data.t0.utc_epoch_milli/1000 + bs(j).data.getX(end));
0153      if dstop < stop
0154        stop = dstop;
0155      end
0156    end
0157
0158
0159    %----------------- Truncate all vectors
0160    utils.helper.msg(msg.PROC1, 'truncate all vectors');
0161    utils.helper.msg(msg.PROC2, 'truncating vectors on interval [%g,%g]', start, stop);
0162
```

```
  % split each ao
  bs = split(bs, plist('split_type', 'times', 'times', [start stop]));

  nsecs = [];
  for j=1:na
    if isempty(nsecs)
      nsecs = bs(j).data.nsecs;
    end
    if nsecs ~= bs(j).data.nsecs
      error('### Something went wrong with the truncation. Vectors don''t span the same time period.');
    end
  end

  %---------------- Set history on output AOs

  for j=1:na
    bs(j).addHistory(getInfo, pl, ao_invars(j), inhists(j));
    bs(j).setName(sprintf('%s(%s)', mfilename, ao_invars{j}), 'internal');
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: consolidate.m,v 1.14 2008/09/05 11:05:28 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('fs', []);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

conv

PURPOSE ^

CONV vector convolution.

SYNOPSIS ^

function varargout = conv(varargin)

DESCRIPTION ^

```
 CONV vector convolution.
%%%
```

```
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 %%%%
```

```
  ii = minfo(mfilename, 'ao', '', 'Signal Processing', '$Id: conv.m,v 1.1 2008/08/21 09:01:03 hewitson Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

SOURCE CODE ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = cpsd(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  if nargout == 0
    error('### cohere cannot be used as a modifier. Please give an output variable.');
  end

  % combine plists
  pl = combine(pl, getDefaultPlist());

  varargout{1} = ao.xspec(as, pl, 'cpsd', getInfo, ao_invars);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: cpsd.m,v 1.9 2008/09/05 11:05:28 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist(...
    'Nfft', -1, ...
    'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
    'Olap', -1, ...
    'Order', 0);
end

```

*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.
%%%
```

```
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 %%%%
```

```
%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: delay.m,v 1.12 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('N', 0, ...
    'method', 'zero');
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = detrend(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Unpack parameter list
  N = find(pl, 'N');

  % Loop over analysis objects
  for jj=1:numel(bs)
    % check data
    if ~isa(bs(jj).data, 'tsdata')
      warning('!!! This method only works with time-series at the moment. Skipping %s', ao_invars{jj});
    else
      % detrend with polynomial
      try
        if find(pl, 'M-FILE ONLY')
          bs(jj).data.setY(polydetrend(bs(jj).data.y.', N));
        else
          bs(jj).data.setY(ltpda_polyreg(bs(jj).data.y, N).');
        end
      catch
        warning('!!! failed to execture ltpda_polyreg.mex. Using m-file call.');
        bs(jj).data.setY(polydetrend(bs(jj).data.y.', N));
      end
      % add history
      bs(jj).addHistory(getInfo, pl, ao_invars(jj), bs(jj).hist);
      % add name
      bs(jj).setName(sprintf('detrend(%s)', bs(jj).name), 'internal');
```

```
    end
  end

  if nargout > 0
    varargout{1} = bs;
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: detrend.m,v 1.10 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('N', 0);
end

function y = polydetrend(varargin)
  % POLYDETREND detrends the input data vector with a polynomial.
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  % DESCRIPTION: POLYDETREND detrends the input data vector with a polynomial.
  %              This function is meant to be used by other ltpda_ functions.
  %              To work with Analysis Objects, use ltpda_polydetrend which calls
  %              this function.
  %
  % CALL:        y = polydetrend(x,order);
  %
  % INPUTS:      x     - vector of x values
  %              order - order of polynomial to fit and subtract
  %
  % OUTPUTS:     y   - detrended data
  %
  % VERSION:     $Id: detrend.m,v 1.10 2008/09/05 11:05:29 ingo Exp $
  %
  % HISTORY:     30-05-07 M Hewitson
  %                 Creation
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  if nargin == 2
    x     = varargin{1};
    order = varargin{2};
    t     = [1:length(x)]';
  elseif nargin == 3
    t     = varargin{1};
    x     = varargin{2};
    order = varargin{3};
  else
    error('### incorrect inputs.');
  end

  % fit polynomial
  p = polyfit(t, x, order);

  % make polynomial series
  py = polyval(p, t);

  % detrend
  y = x - py;
end
% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = dft(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Make copies or handles to inputs
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Loop over input AOs
  for j=1:length(bs)
    if ~isa(bs(j).data, 'tsdata')
      warning('!!! The DFT can only be computed on input time-series. Skipping AO %s', ao_invars{j});
    else

      % Extract necessary parameters
      f = find(pl, 'f');

      % Compute f if necessary
      if f == -1
        f = linspace(0, bs(j).data.fs/2, length(bs(j).data.getY)/2+1);
      end

      % Compute DFT
      fs  = bs(j).data.fs;
      N   = length(bs(j).data.getY);
      J   = -2*pi*1i.*[0:N-1]/fs;
      dft = zeros(size(f));
      for kk = 1:length(f)
        dft(kk) = exp(f(kk)*J)*bs(j).data.getY;
      end

      % Make output fsdata AO
      yunits = bs(j).data.yunits;
      % Keep the data shape of the input AO
      if size(bs(j).data.y,1) ~= 1
        f   = f.';
        dft = dft.';
      end
      bs(j).data = fsdata(f, dft, bs(j).data.fs);
      bs(j).setXunits('Hz', 'internal');
      bs(j).setYunits(yunits./unit('Hz'), 'internal');
      % Set name
      bs(j).setName(sprintf('dft(%s)', ao_invars{j}), 'internal');
```

```
      % Add history
      bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    end
  end

  % Set outputs
  if nargout > 0
    varargout{1} = bs;
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: dft.m,v 1.13 2008/09/05 14:14:11 hewitson Exp $', sets, pls);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('f', -1);
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

diff

PURPOSE ^

DIFF differentiates the data in AO.

SYNOPSIS ^

function varargout = diff(varargin)

DESCRIPTION ^

```
 DIFF differentiates the data in AO.
%%%
```

- 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 ^

```
% 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
%
%%%
```

```
        % y'(xm)
        z(m) = (1/dx(m-2)+1/dx(m-1))*(y(m)-y(m-1))+...
          dx(m-1)/(dx(m-1)*dx(m-2)+dx(m-2)^2)*(y(m-2)-y(m));
        % y'(xi) (i>1 & i<m)
        dx1 = repmat(dx(1:m-2),1,1);
        dx2 = repmat(dx(2:m-1),1,1);
        y1 = y(1:m-2); y2 = y(2:m-1); y3 = y(3:m);
        z(2:m-1) = 1./(dx1.*dx2.*(dx1+dx2)).*...
          (-dx2.^2.*y1+(dx2.^2-dx1.^2).*y2+dx1.^2.*y3);
        bs(j).setY(z, 'internal');
      case 'ORDER2SMOOTH'
        x  = bs(j).data.getX;
        y  = bs(j).data.getY;
        dx = diff(x);
        m  = length(y);
        if max(abs(diff(dx)))>sqrt(eps(max(abs(dx))))
          error('### The x-step must be constant for method ''ORDER2SMOOTH''')
        elseif m<5
          error('### Length of y must be at least 5 for method ''ORDER2SMOOTH''.')
        end
        h = mean(dx);
        z = zeros(size(y));
        % y'(x1)
        z(1) = sum(y(1:5).*[-54; 13; 40; 27; -26])/70/h;
        % y'(x2)
        z(2) = sum(y(1:5).*[-34; 3; 20; 17; -6])/70/h;
        % y'(x{m-1})
        z(m-1) = sum(y(end-4:end).*[6; -17; -20; -3; 34])/70/h;
        % y'(xm)
        z(m) = sum(y(end-4:end).*[26; -27; -40; -13; 54])/70/h;
        % y'(xi) (i>2 & i<(N-1))
        Dc = [2 1 0 -1 -2];
        tmp = convn(Dc,y)/10/h;
        z(3:m-2) = tmp(5:m);
        bs(j).setY(z, 'internal');
      case 'filter'
      otherwise
        error('### Unknown method for computing the derivative.');
    end

    % add history
    bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    % name for this object
    bs(j).setName(sprintf('diff(%s)', ao_invars{j}), 'internal');
  end

  % Set outputs
  if nargout > 0
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: diff.m,v 1.10
2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('method',  '3POINT');
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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:
%
%%%
```

```
0091
0092   %%% Reshape the ouput to the same size of the input
0093   varargout{1} = bs;
0094
0095 end
0096
0097 %%%%
```

dropduplicates

PURPOSE ^

DROPDUPLICATES drops all duplicate samples in time-series AOs.

SYNOPSIS ^

function varargout = dropduplicates(varargin)

DESCRIPTION ^

```
 DROPDUPLICATES drops all duplicate samples in time-series AOs.
%%%
```

```
% DROPDUPLICATES drops all duplicate samples in time-series AOs.
%%%
```

```
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: dropduplicates.m,v 1.10 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('tol', 5e-3);
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%
```

```
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%
%%%
```

```
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

filter

PURPOSE ^

FILTER overrides the filter function for analysis objects.

SYNOPSIS ^

function varargout = filter(varargin)

DESCRIPTION ^

```
 FILTER overrides the filter function for analysis objects.
%%%
```

SUBFUNCTIONS ^

- function ii = getInfo(varargin)
- function pl_default = getDefaultPlist()

SOURCE CODE ^

```
% FILTER overrides the filter function for analysis objects.
%%%
```

```
  if ~isa(fobj, 'miir') && ~isa(fobj, 'mfir')
    error('### the filter input should be an miir/mfir object.');
  end

  % Now copy the filter so we can change it
  fobj = copy(fobj, 1);

  for j=1:numel(bs)
    %--------- Time-series filter
    if isa(bs(j).data, 'tsdata')
      % get input data
      if isa(fobj, 'mfir')
        % apply filter
        utils.helper.msg(msg.PROC1, 'filtering with FIR filter');
        [bs(j).data.y, Zf] = filter(fobj.a, 1, bs(j).data.y, fobj.histout);
        fobj.setHistout(Zf);
        % remove group delay
        if isempty(find(pl, 'gdoff'))
          gd = floor(fobj.gd);
          bs(j).data.setX(bs(j).data.getX(1:end-gd));
          bs(j).data.setY(bs(j).data.y(1+gd:end));
          bs(j).data.collapseX;
        end
      else %if isa(fobj, 'miir')
        if bs(j).data.fs ~= fobj.fs
          warning('!!! Filter is designed for a different sample rate of data.');
          % Adjust/redesign if this is a standard filter
          fobj = fobj.redesign(bs(j).data.fs);
        end
        utils.helper.msg(msg.PROC1, 'filtering with IIR filter');
        [bs(j).data.y, Zf] = filter(fobj.a, fobj.b, bs(j).data.y, fobj.histout);
        fobj.setHistout(Zf);
      end
      % add history
      bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
      % name for this object
      bs(j).setName(sprintf('%s(%s)', fobj.name, ao_invars{j}), 'internal');

    elseif isa(d, 'fsdata')
      %--------- Frequency-series filter
      error('### I don''t work yet. Please code me up.');
    else
      error('### unknown data type.');
    end
  end

  % Set outputs
  if nargout == 1
    varargout{1} = bs;
  elseif nargout == 2
    varargout{1} = bs;
    varargout{2} = fobj;
  elseif nargout > 2
    error('### wrong number of output arguments.');
  end
end

%------------------------------------------------------------------------
% Get Info Object
%------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  else
    sets = {'Default'};
    pls  = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: filter.m,v 1.40 2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%------------------------------------------------------------------------
% Get Default Plist
%------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('filter',  '', 'gdoff' ,[]);
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = filtfilt(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);
  [fobj, f_invars] = utils.helper.collect_objects(varargin(:), 'ltpda_filter',
in_names);

  % Make copies or handles to inputs
  bs   = copy(as, nargout);

  % combine plists
  pl = combine(pl, getDefaultPlist());

  if isempty(fobj)
    fobj         = find(pl, 'filter');
    f_invars{1} = class(fobj);
  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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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/smoother)
            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 ^

```
% 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/smoother)
%            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
%
%%%
```

```
    varargout{1} = bs;
  end
  if nargout > 1
    varargout{2} = filts;
  end
  if nargout > 2
    varargout{3} = nfs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: firwhiten.m,v 1.10 2008/09/08 08:30:12 hewitson Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist(...
    'Nfft', -1, ...
    'bw', 20, ...
    'hc', 0.8, ...
    'Win', specwin('Kaiser', 10, 150), ...
    'order', 0, ...
    'FIRwin', specwin('Hanning', 10), ...
    'Ntaps', 256);
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 $
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = fixfs(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao invars] = utils.helper.collect objects(varargin(:), 'ao', in names);
```

```
  [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % combine plists
  pl = combine(pl, getDefaultPlist());

  % Get fs and t0
  t0s    = find(pl, 't0');
  fss    = find(pl, 'fs');
  method = find(pl, 'method');

  if numel(fss) ~= 1 && numel(fss) < na
    error('### Please specify either a no sample rate, a single sample rate, or one for each input time-series.');
  end

  % Get only tsdata AOs
  for j=1:numel(bs)
    if isa(bs(j).data, 'tsdata')
      % record input hist
      hin = bs(j).hist;
      utils.helper.msg(msg.PROC1, 'fixing AO: %s', bs(j).name);
      %------------- Fit sample rate and t0
      [ffs, ft0] = tsdata.fitfs(bs(j).data.getX);
      %---------------- Get target sample rate
      if numel(fss) > 1
        fs = fss(j);
      else
        fs = fss;
      end
      if fs < 0
        utils.helper.msg(msg.PROC1, 'using sample rate from fit: %f', ffs);
        fs = ffs;
      end
      %---------------- Get target start time
      if numel(t0s) > 1
        t0 = t0s(j);
      else
        t0 = t0s;
      end
      if t0 < 0
        utils.helper.msg(msg.PROC1, 'using start time from fit: %f', ft0);
        t0 = ft0;
      end
      t0 = t0 + bs(j).data.getX(1);
      %-------------- Compute new grid
      switch method
        case 'Samples'
          N = length(bs(j).data.y);
          t = linspace(t0, (N-1)/fs, N);
        case 'Time'
          Nsecs = bs(j).data.nsecs;
          t = t0 + [0:1/fs:Nsecs-1/fs].';
        otherwise
          error('### Unknown interpolation method. Do you want to preserve data duration or number of samples?');
      end
      %-------------- Interpolate
      interp(bs(j), plist('vertices', t));
      % Add history
      bs(j).addHistory(getInfo, pl, ao_invars(j), hin);
      % Set name
      bs(j).setName(sprintf('%s(%s)', mfilename, ao_invars{j}), 'internal');
    else
      warning('!!! Skipping AO %s - it''s not a time-series AO.', ao_invars{j});
      bs(j) = [];
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: fixfs.m,v 1.11 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('fs', -1, 't0', -1, 'Method', 'Time');
end


```

*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.
```

```
%%%
```

```
      % Prepare for generation
      rphi = zeros(N,1);                    % Empty vector for random phases
      xs   = zeros(fs*(cNsecs+segLen), 1);  % Large empty vector for new time-series
      e1   = 1; e2 = segLen*fs;             % Indices into large vector
      step = round(segLen*fs*Olap);         % step size between each new segment
      lxs  = length(xs);

      % Loop over segments
      for s=1:Nsegs
        % Generate random phase vector
        rphi(2:N/2) = pi*rand(1,N/2-1);  % First half
        rphi(N/2+1) = pi*round(rand);    % mid point
        rphi(N/2+2:N) = -rphi(N/2:-1:2); % reflected half
        %---- Compute Fourier amplitudes
        % Use chi^2 distribution to randomize amplitudes.
        % - from Percival and Walden: S_est = S.*chi2rnd(2)/2
        %   so A_est = A.*sqrt(chi2rnd(2)/2)
        % Here we take the measured input data to be a good estimate of
        % the underlying power spectrum
        X = (Ak.*sqrt(chi2rnd(2)/2)) .*exp(1i.*rphi);
        % Inverse FFT
        x  = ifft(X, 'symmetric');
        % overlap the segments
        xs(e1:e2) = xs(e1:e2) + win.*x;
        % increase step
        e1 = e1 + step;
        e2 = e2 + step;
        if e2>lxs
          break
        end
      end
      % Make ao from the segment of data we want
      e1 = fs*segLen/2;
      e2 = fs*(Nsecs+segLen/2)-1;
      b  = ao(tsdata(xs(e1:e2).', fs));
      b.setName(sprintf('fngen(%s)', ao_invars{j}), 'internal');
      b.setXunits('s', 'internal');
      % Add history
      b.addHistory(getInfo, pl, ao_invars(j), as(j).hist);
      % Add to outputs
      bs = [bs b];
    end
  end

  % Set outputs
  varargout{1} = bs;
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: fngen.m,v 1.17 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('Nsecs', -1, 'Win', getappdata(0, 'ltpda_default_spectral_window'));
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

gapfilling

PURPOSE ^

GAPFILLING fills possible gaps in data.

SYNOPSIS ^

function varargout = gapfilling(varargin)

DESCRIPTION ^

```
 GAPFILLING fills possible gaps in data.
%%%
```

- function plf = getlpFilter(x)
- function phf = gethpFilter(x)
- function ii = getInfo(varargin)
- function plo = getDefaultPlist()

SOURCE CODE ^

```
% GAPFILLING fills possible gaps in data.
%%%
```

```
data

%----------------------------
% Get data type from the first AO
a1    = as(1).data;
dinfo1 = whos('a1');
dtype1 = dinfo1.class;

% Get data type from the second AO
a2     = as(2).data;
dinfo2 = whos('a2');
dtype2 = dinfo2.class;

if a1.fs ~= a2.fs
  warning('Sampling frequencies of the two AOs are different. The sampling frequency of the first AO will be used to reconstruct the gap.')
end

% Different units error
if a1.xunits ~= a2.xunits
  error('!!! Data has different X units !!!');
end

if a1.yunits ~= a2.yunits
  error('!!! Data has different Y units !!!');
end

a1_length = length(a1.getX);
a2_length = length(a2.getX);
gaptime   = a2.getX(1) - a1.getX(a1_length);
gapn      = gaptime*a1.fs-1;
t         = (0:1:gapn-1)'/a1.fs;

%--- gapfilling process itself
if strcmp(method,'linear')
  % linear interpolation method ---xfilled=(deltay/deltax)*t+y1(length(y1))---
  if length(a1.getY)>10 && length(a2.getY)>10
    dy = mean(a2.getY(1:10))-mean(a1.getY(a1_length-10:a1_length));

    filling_data = (dy/gaptime)*t + mean(a1.getY(a1_length-10:a1_length));
    filling_time = (1:1:gapn)'/a1.fs + a1.getX(a1_length);
  else
    error('!!! Not enough data in the data segments (min=11 for each one for the linear method) !!!');
  end

elseif strcmp(method,'spline') % spline method xfilled = a*T^3 + b*T^2 + c*T +d

  if length(a1.getY)>1000 && length(a2.getY)>1000

    % derivatives of the input data are calculated
    da1 = diff(a1.getY(1:100:a1_length))*(a1.fs/100);
    da1 = tsdata(da1, a1.fs/100);
    da1 = ao(da1);

    da2 = diff(a2.getY(1:100:a2_length))*(a2.fs/100);
    da2 = tsdata(da2, a2.fs/100);
    da2 = ao(da2);

    % This filters the previous derivatives
    % filters parameters are obtained
    plfa1 = getlpFilter(a1.fs/100);
    plfa2 = getlpFilter(a2.fs/100);

    lpfa1 = miir(plfa1);
    lpfpla1 = plist(param('filter', lpfa1));

    lpfa2 = miir(plfa2);
    lpfpla2 = plist(param('filter', lpfa2));

    % derivatives are low-pass filtered
    da1filtered = filtfilt(da1, lpfpla1);
    da2filtered = filtfilt(da2, lpfpla2);

    % coefficients are calculated
    c = mean(da1filtered.data.getY(length(da1filtered.data.getX)...
      -10:length(da1filtered.data.getX)));
    d = mean(a1.getY(length(a1.getY)-10:length(a1.getY)));

    a=(2*d+(c+mean(da2filtered.data.getY(1:10)))...
      *gaptime-2*mean(a2.getY(1:10)))/(gaptime.^3);

    b=-(3*d+2*c*gaptime+mean(da2filtered.data.getY(1:10))...
      *gaptime-3*mean(a2.getY(1:10)))/(gaptime^2);

    % filling data is calculated with the coefficients a, b, c and d
```

```
      filling_data = a*t.^3+b*t.^2+c*t+d;
      filling_time = (1:1:gapn)/a1.fs + a1.getX(a1_length);
    else
      error('!!! Not enough data in data segments (min=1001 in spline method)');
    end

  end

  % this add noise (if desired) to the filled gap
  if strcmp(addnoise,'yes');
    % calculation of the standard deviation after eliminating the low-frequency component
    phpf = gethpFilter(a1.fs);
    ax = tsdata(a1.getY, a1.fs);
    ax = ao(ax);
    hpf = miir(phpf);
    hpfpl = plist(param('filter', hpf));
    xhpf = filter(ax, hpfpl);
    hfnoise = std(xhpf);

    % noise is added to the filling data
    filling_data = filling_data + randn(length(filling_data),1)*hfnoise.data.getY;
  end

  % join data
  filling_data = [a1.getY; filling_data; a2.getY];
  filling_time = [a1.getX; filling_time; a2.getX];

  % create new output tsdata
  ts = tsdata(filling_time, filling_data);
  ts.setYunits(a1.yunits);
  ts.setXunits(a1.xunits);

  % create new output history
  %%histin = [as(1).hist as(2).hist];
  %%h = history(ALGONAME, VERSION, pls, histin);
  %%h = set(h, 'invars', invars);

  % make output analysis object
  b = ao(ts);
  b.setName(sprintf('gapfilling(%s&%s)', ao_invars{1}, ao_invars{2}), 'internal');
  %warning('%%%
```

```
%                 Creation.
%
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = hist(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pli, pl_invars, rest] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  pl = combine(pli, getDefaultPlist('Number of bins'));

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Get parameters
  N = find(pl, 'N');
  X = find(pl, 'X');

  %---------------- Loop over input AOs

  % start looping
  for j=1:numel(bs)
    % Histogram this data
    if isempty(X)
      [n,x] = hist(bs(j).data.y, N);
    else
      [n,x] = hist(bs(j).data.y, X);
    end
    % Keep the data shape of the input AO
    if size(bs(j).data.y, 1) ~= 1
```

```
      x = x.';
      n = n.';
    end
    % make a new xydata object
    xy = xydata(x, n);
    xy.setXunits(bs(j).data.yunits);
    xy.setYunits('Count');
    % make output analysis object
    bs(j).data = xy;
    % name for this object
    bs(j).setName(sprintf('hist(%s)', ao_invars{j}), 'internal');
    % Add history
    bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    % Add to outputs
  end % end of AO loop

  % Set outputs
  varargout{1} = bs;
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Number of bins', 'Bin centres'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: hist.m,v 1.23
2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------

function plo = getDefaultPlist(set)
  switch set
    case 'Number of bins'
      plo = plist('N', 10);
    case 'Bin centres'
      plo = plist('X', []);
    otherwise
      error('### Unknown default plist for the set [%s]', set);
  end
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ifft

PURPOSE ^

IFFT overloads the ifft operator for Analysis objects.

SYNOPSIS ^

function varargout = ifft(varargin)

DESCRIPTION ^

```
 IFFT overloads the ifft operator for Analysis objects.
%%%
```

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ifft(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % combine plists
  pl = combine(pl, getDefaultPlist());

  % two-sided or one?
  type    = find(pl, 'type');
  twoside = find(pl, 'twoside');

  % Check input analysis object
  for j=1:numel(bs)
    switch class(bs(j).data)
      case {'fsdata'}
        if strcmpi(twoside, 'YES')
          y = [bs(j).data.getY; bs(j).data.getY(end-1:-1:2)];
        else
          y = bs(j).data.y;
        end
        % make a new fsdata object
        y    = ifft(y, type);
        % compute time axis
        N   = length(y);
        x   = linspace(0, (N-1)/bs(j).data.fs, N);
        % Keep the data shape if the input AO
        if size(bs(j).data.y,1) == 1
          x = x.';
          y = y.';
        end
        fsd = tsdata(x, y);
        fsd.setXunits('s');
        fsd.setYunits('arb');

        % Set data
        bs(j).data = fsd;
        % Set name
        bs(j).setName(sprintf('ifft(%s)', ao_invars{j}), 'internal');
        % Add history
        bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);

      case {'tsdata', 'cdata', 'xydata'}
        error('### I don''t work for time-series, constant or x/y data.');
```

```
      otherwise
        error('### unknown data type.')
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: ifft.m,v 1.11 2008/09/08 08:30:56 hewitson Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('type', 'symmetric', 'twoside', 'yes');
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

interpmissing

PURPOSE ^

INTERPMISSING interpolate missing samples in a time-series.

SYNOPSIS ^

function varargout = interpmissing(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 = interpmissing(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('interpmissing')

              Get information about a specified set-plist by calling:
              >> ao.getInfo('interpmissing', 'None')

 VERSION:      $Id: interpmissing.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 ^

```
% 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 = interpmissing(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('interpmissing')
%
%               Get information about a specified set-plist by calling:
%               >> ao.getInfo('interpmissing', 'None')
%
% VERSION:      $Id: interpmissing.m,v 1.12 2008/09/08 08:31:18 hewitson Exp $
%
% HISTORY:      22-05-08 M Hewitson
%                  Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = interpmissing(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Get tolerance
  dtol = find(pl, 'd');

  % Get only tsdata AOs
  for j=1:numel(bs)
    if isa(bs(j).data, 'tsdata')

      % capture input history
      ih = bs(j).hist;

      % find missing samples
      t    = [];
      d    = diff(bs(j).data.getX);
      idxs = find(d>dtol/bs(j).data.fs);
      utils.helper.msg(msg.PROC1, 'found %d data gaps', numel(idxs));

      % create new time grid
      count = 0;
      for k=1:numel(idxs)
        idx = idxs(k);
        if isempty(t)
          t   = bs(j).data.getX(1+idx-idxs(1):idx);
```

```
        end
        % now add samples at 1/fs until we are within 1/fs of the next sample
        while bs(j).data.getX(idx+1)-t(end) > 1/bs(j).data.fs
          t(end+1) = t(end)+1/bs(j).data.fs;
          count = count + 1;
        end
        if k==numel(idxs)
          t = [t; bs(j).data.getX(idx+1:end)];
        else
          t = [t; bs(j).data.getX(idx+1:idxs(k+1))];
        end
      end
      utils.helper.msg(msg.PROC1, 'filled with %d samples', count);

      % now interpolate onto this new time-grid
      if ~isempty(t)
        interp(bs(j), plist('vertices', t));
        bs(j).setName(sprintf('interpmissing(%s)', ao_invars{j}), 'internal');
        % Add history
        bs(j).addHistory(getInfo, pl, ao_invars(j), ih);
      else
        utils.helper.msg(msg.PROC1, 'no missing samples found in %s - no action performed.', ao_invars{j});
      end
    else
      utils.helper.msg(msg.PROC1, 'skipping AO %s - it''s not a time-series AO.', ao_invars{j});
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: interpmissing.m,v 1.12 2008/09/08 08:31:18 hewitson Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('d', 1.5);
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 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('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 ^

```
% 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 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('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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = lcohere(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Compute coherence with lxspec
  varargout{1} = ao.lxspec(as, pl, 'cohere', getInfo, ao_invars);

end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: lcohere.m,v 1.7 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('Kdes', 100, 'Jdes', 1000, 'Lmin', 0,...
    'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
    'Olap', -1, 'Order', 0);
end

```

*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 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('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 ^

```
% LCPSD implement cross-power-spectral density estimation computed on a log frequency axis.
%%%
```

```
%
%%%
```

lincom

PURPOSE ^

LINCOM

SYNOPSIS ^

function varargout = lincom(varargin)

DESCRIPTION ^

```
 LINCOM
%%%
```

```
%%%
```

```
  for kk = 2:numel(ts);
    new_units = new_units + ts(kk).data.yunits;
  end

  % Make linear combination and collect name string
  nstr = '';
  y = zeros(size(ts(1).data.y));
  for j=1:nc
    nstr = [nstr num2str(coeffs(j)) '*' ao_invars{j} ' + '];
    y = y + coeffs(j).*ts(j).data.y;
  end
  nstr = deblank(nstr(1:end-2));

  % create new ao
  b = ao;
  b.data = copy(ts(1).data,1);
  b.data.setY(y);

  %%% Set Name
  b.setName(nstr, 'internal');

  %%% Set new Units
  b.setYunits(new_units, 'internal');

  %%% Add History
  b.addHistory(getInfo('None'), pl, ao_invars, [ts.hist chist]);

  %%% Reshape the ouput to the same size of the input
  varargout{1} = b;
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, 'ao', '', utils.const.categories.sigproc, '$Id: lincom.m,v 1.9
2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

SOURCE CODE ^

```
% LINEDETECT find spectral lines in the ao/fsdata objects.
%%%
```

```
      f = [lines(:).f];
      y = [lines(:).a];

      % Keep the data shpare of the input AO
      if size(bs(j).data.y, 2) == 1
        f = f.';
        y = y.';
      end

      % Make output data
      fs = fsdata(f, y, bs(j).data.fs);

    else
      error('### I can only find lines in frequency-series AOs.');
    end

    %------- Make output AO

    % make output analysis object
    bs(j).data = fs;

    bs(j).setName(sprintf('lines(%s)', ao_invars{1}));
    bs(j).addHistory(getInfo('None'), pl, ao_invars(j), bs(j).hist);
  end

  % return output
  varargout{1} = bs;
end

%-------------------------------------------------------------------------
% find spectral lines
function lines = findLines(ay, f, nay, thresh, N, fsearch)

  % look for spectral lines
  l       = 0;
  pmax    = 0;
  pmaxi   = 0;
  line    = [];
  idx     = find( f>=fsearch(1) & f<=fsearch(2) );
  for j=1:length(idx)
    v = nay(idx(j));
    if v > thresh
      if v > pmax
        pmax  = v;
        pmaxi = idx(j);
      end
    else
      if pmax > 0
        % find index when we have pmax
        fidx = pmaxi; %(find(nay(1:idx(j))==pmax));
        l = l+1;
        line(l).idx = fidx;
        line(l).f   = f(fidx);
        line(l).a   = ay(fidx);
      end
      pmax = 0;
    end
  end

  % Select largest peaks
  lines = [];
  if ~isempty(line)
    [bl, lidx] = sort([line.a], 'descend');
    lidxs = lidx(1:min([N length(lidx)]));
    lines = line(lidxs);
    disp(sprintf('   + found %d lines.', length([lines.f])));
  else
    disp('   + found 0 lines.');
  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, 'm', '', utils.const.categories.sigproc, '$Id: linedetect.m,v 1.3 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%%%
```

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 ^

```
% 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.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = lpsd(varargin)

  import utils.const.*

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

```

```
% Decide on a deep copy or a modify
bs = copy(as, nargout);

% Combine plists
pl = combine(pl, getDefaultPlist);

% Go through each input AO
for j=1:numel(bs)
  % check this is a time-series object
  if ~isa(bs(j).data, 'tsdata')
    warning('!!! lpsd requires tsdata (time-series) inputs. Skipping AO %s', ao_invars{j});
  else
    % Desired number of averages
    Kdes = find(pl, 'Kdes');
    % num desired spectral frequencies
    Jdes = find(pl, 'Jdes');
    % Window function
    Win = find(pl, 'Win');
    % Overlap
    Nolap = find(pl, 'Olap');
    if Nolap < 0
      % use window function rov
      Nolap = Win.rov/100;
      utils.helper.msg(msg.PROC1, 'using recommended overlap of %d', Nolap);
      pl.pset('Olap', Nolap);
    end
    % Order of detrending
    Order = find(pl, 'Order');
    % Minimum segment length
    Lmin = find(pl, 'Lmin');
    % Get frequency vector
    [f, r, m, L, K] = ao.ltf_plan(length(bs(j).data.y), bs(j).data.fs, Nolap, 1, Lmin, Jdes, Kdes);
    % compute LPSD
    try
      if find(pl, 'M-FILE ONLY')
        % Using pure m-file version
        [P, Pxx, ENBW] = ao.mlpsd_m(bs(j).data.y, f, r, m, L, bs(j).data.fs, Win, Order, Nolap);
      else
        [P, Pxx, ENBW] = ao.mlpsd_mex(bs(j).data.y, f, r, m, L, bs(j).data.fs, Win, Order, Win.rov, Lmin);
      end
    catch
      warning('!!! mex file dft failed. Using m-file version of lpsd.');
      % Using pure m-file version
      [P, Pxx, ENBW] = ao.mlpsd_m(bs(j).data.y, f, r, m, L, bs(j).data.fs, Win, Order, Nolap);
    end

    % Keep the data shape of the input AO
    if size(bs(j).data.y,1) == 1
      P   = P.';
      Pxx = Pxx.';
      f   = f.';
    end

    % create new output fsdata
    scale = find(pl, 'Scale');
    switch scale
      case 'AS'
        fsd = fsdata(f, sqrt(P), bs(j).data.fs);
        fsd.setYunits(bs(j).data.yunits);
      case 'ASD'
        fsd = fsdata(f, sqrt(Pxx), bs(j).data.fs);
        fsd.setYunits(bs(j).data.yunits / unit('Hz^0.5'));
      case 'PS'
        fsd = fsdata(f, P, bs(j).data.fs);
        fsd.setYunits(bs(j).data.yunits.^2);
      case 'PSD'
        fsd = fsdata(f, Pxx, bs(j).data.fs);
        fsd.setYunits(bs(j).data.yunits.^2/unit('Hz'));
      otherwise
        error(['### Unknown scaling:' scale]);
    end
    fsd.setXunits('Hz');
    fsd.setEnbw(ENBW);
    % make output analysis object
    bs(j).data = fsd;
    % set name
    bs(j).setName(sprintf('l%s(%s)', scale, ao_invars{j}), 'internal');
    % Add history
    bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    % Add processing info
    bs(j).procinfo = plist('r', r, 'm', m, 'l', L, 'k', K);
```

```
    end % End tsdata if/else
  end % End AO loop

  % Set output
  varargout{1} = bs;

end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', 'Signal Processing', '$Id: lpsd.m,v 1.13 2008/09/05 14:14:58 hewitson Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('Kdes', 100, 'Jdes', 1000, 'Lmin', 0,...
    'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
    'Olap', -1, 'Order', 0, 'Scale', 'PSD');
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltfe(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pl              = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Combine plists
  pl = combine(pl, getDefaultPlist);

  % Compute coherence with lxspec
  varargout{1} = ao.lxspec(as, pl, 'tfe', getInfo, ao_invars);
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, '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, 'ltpda_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
%%%
```

- 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 ^

```
% PSD makes power spectral density estimates of the time-series objects
%%%
```

```
  pl = combine(pl, getDefaultPlist());

  % Loop over input AOs
  for j=1:numel(bs)
    % check input data
    if isa(bs(j).data, 'tsdata')
      fs = bs(j).data.fs;
      %--------- Check all defaults are sensible
      % Check the number of points in FFT. If this is not set (<0) we set it
      % to be the length of the input data.
      Nfft = find(pl, 'Nfft');
      setWindow = 0;
      if ischar(Nfft)
        nNfft = floor(eval(Nfft));
        utils.helper.msg(msg.PROC1, 'setting Nfft to %s = %d', Nfft, nNfft);
        Nfft = nNfft;
      else
        if Nfft <= 0
          Nfft = length(bs(j).data.y);
          utils.helper.msg(msg.PROC1, 'using default Nfft of %g', Nfft);
          setWindow = 1;
        end
      end
      pl.pset('Nfft', Nfft);

      % Check the window function. If the length of the window doesn't match
      % NFFT then we resize it.
      Win = find(pl, 'Win');   % Window object
      if setWindow || length(Win.win)~=Nfft
        switch Win.type
          case {'Kaiser', 'Flattop'}
            Win = specwin(Win.type, Nfft, Win.psll);
          otherwise
            Win = specwin(Win.type, Nfft);
        end
        utils.helper.msg(msg.PROC1, 'reset window to %s(%d)', strrep(Win.type, '_', '\_'), length(Win.win));
      end
      pl.pset('Win', Win);

      % Check the overlap. If this is not set, we take the overlap from that
      % recommended by the window function.
      Olap = find(pl, 'Olap');
      if Olap <= 0
        Olap = Win.rov;
        utils.helper.msg(msg.PROC1, 'using default overlap of %2.1f%%', Olap);
      end
      pl.pset('Olap', Olap);

      % scaling of spectrum
      scale = find(pl, 'Scale');

      % check parameters
      if length(Win.win) ~= Nfft
        error('### Window length should be same as Nfft.');
      end
      % Compute PSD using pwelch
      [pxx, f, info] = ao.welch(bs(j), 'psd', pl);
      % Keep the data shape of the input AO
      if size(bs(j).data.y,1) == 1
        pxx = pxx.';
        f   = f.';
      end
      % create new output fsdata
      fsd = fsdata(f, pxx, fs);
      fsd.setXunits('Hz');
      fsd.setYunits(info.units);
      fsd.setEnbw(Win.nenbw*fs/Nfft);
      fsd.setNavs(info.navs);
      fsd.setT0(bs(j).data.t0);
      % update AO
      bs(j).data = fsd;
      % set name
      bs(j).setName(sprintf('%s(%s)', scale, ao_invars{j}), 'internal');
      % Add history
      bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    else
      warning('### Ignoring input AO number %d (%s); it is not a time-series.', j, bs(j).name)
    end
  end % loop over analysis objects

  % set output
  varargout{1} = bs;
end
```

```
%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: psd.m,v 1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist(...
    'Nfft', -1, ...
    'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
    'Olap', -1, ...
    'Scale', 'PSD', ...
    'Order', 0);
end
% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 AO.

              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 ^

```
% 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 AO.
%
%              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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = pwelch(varargin)
  warning('! This method is deprecated and will be removed from future releases of the toolbox. Use ao/psd instead.')
  if nargout == 0
    psd(varargin{:})
  else
    varargout{:} = psd(varargin{:});
  end
end

% END
```

rms

PURPOSE ^

RMS Calculate RMS deviation from spectrum

SYNOPSIS ^

function varargout = rms(varargin)

DESCRIPTION ^

```
 RMS Calculate RMS deviation from 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = rms(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % Loop over input AOs
  for j=1:numel(bs)
    % check input data
    if isa(bs(j).data, 'fsdata')
      % get data
      f = bs(j).data.getX;
      spe = [f.' bs(j).data.getY.'];
      % start and end frequencies
      s = f(1);
      e = f(end);
      % compute integrated rms
      l1=spe(:,1)>=s;
      sp=spe(l1,:);
      l2=sp(:,1)<=e;
      sp=sp(l2,:);
      si=size(sp);
      li=si(1,1);
      freq=sp(:,1);
      sp2=sp(:,2).^2;
      ms=sp2;
      for i= li-1 :-1: 1
        ms(i)=ms(i+1)+(sp2(i+1)+sp2(i))*(freq(i+1)-freq(i))/2;
      end
      % set data
      bs(j).setXY(freq, sqrt(ms), 'internal');
      % set name
      bs(j).setName(sprintf('RMS(%s)', ao_invars{j}), 'internal');
      % Add history
      bs(j).addHistory(getInfo, plist, ao_invars(j), bs(j).hist);
    else
      warning('!!! Skipping AO %s - it''s not an frequency series.', ao_invars{j});
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = bs;
  end
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  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:

- lxspec LXSPEC performs log-scale cross-spectral analysis of various forms.

SUBFUNCTIONS ^

- function ii = getInfo(varargin)
- function pl_default = getDefaultPlist()

SOURCE CODE ^

Description of select

```
% SELECT select particular samples from the input AOs and return new AOs with only those samples.
%%%
```

```
    varargout{1} = bs;
  end

end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: select.m,v 1.26 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function pl_default = getDefaultPlist()
  pl_default = plist('samples', []);
end

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = smoother(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs and plists
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [pl, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Decide on a deep copy or a modify
  bs = copy(as, nargout);

  % combine plists
  pl = combine(pl, getDefaultPlist());

  % Get parameters from plist
  bw      = find(pl, 'width');
  hc      = find(pl, 'hc');
  method  = find(pl, 'method');

  % check the method
  if  ~strcmp(method, 'median') && ...
      ~strcmp(method, 'mean') && ...
      ~strcmp(method, 'min') && ...
      ~strcmp(method, 'max') && ...
      ~strcmp(method, 'mode')
    help(mfilename)
    error('### Unknown smoothing method');
  end

  % Loop over input AOs
  for j=1:length(bs)
    utils.helper.msg(msg.PROC1, 'smoothing %s', bs(j).name);
    if strcmp(method, 'median')
      bs(j).data.setY(ltpda_smoother(bs(j).data.getY, bw, hc, method), 'internal');
    else
      bs(j).data.setY(smooth(bs(j).data.getY, bw, hc, method), 'internal');
    end
    % Add history
    bs(j).addHistory(getInfo, pl, ao_invars(j), bs(j).hist);
    % set name
    bs(j).setName(sprintf('smoother(%s)', ao_invars{j}), 'internal');
  end

  if nargout > 0
    varargout{1} = bs;
  end
end
```

```
%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: smoother.m,v 1.12 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function pl = getDefaultPlist()
  pl = plist('width', 20, 'hc', 0.8, 'method', 'median');
end

%-------------------------------------------------------------------------
% smooth data
function ys = smooth(y, bw, hc, method)
  N = length(y);
  ys = zeros(size(y));
  for kk=1:N
    if mod(kk, 1000)==0
      disp(sprintf(' -- smoothed %06d samples', kk));
    end
    % Determine the interval we are looking in
    interval = kk-bw/2:kk+bw/2;
    interval(interval<=0)=1;
    interval(interval>N)=N;
    % calculate median value of interval
    % after throwing away outliers
    trial = sort(y(interval));
    b = round(hc*length(trial));
    eval(sprintf('ys(kk) = %s(trial(1:b));', method));
  end
end
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = spectrogram(varargin)

  bs = [];

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  if nargout == 0
    error('### spectrogram cannot be used as a modifier. Please give an output variable.');
  end

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  [ps, pl_invars] = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % Process parameters
  pl = combine(ps, getDefaultPlist);

  % Check input analysis object
  for j=1:numel(as)
    a     = as(j);

    % Get settings for this AO
    nfft = find(pl, 'Nfft');
    if isempty(nfft) || nfft < 0
      nfft = a.data.fs;
    end
    win  = find(pl, 'Win');
    if length(win.win) < nfft
      switch win.type
        case {'Kaiser', 'Flattop'}
          win = specwin(win.type, nfft, win.psll);
        otherwise
          win = specwin(win.type, nfft);
      end
      utils.helper.msg(msg.PROC1, 'reset window to %s(%d)', strrep(win.type, '_', '\_'), length(win.win));
    end
    nolap = find(pl, 'Nolap');
    if isempty(nolap) || nolap < 0
      nolap = floor(win.rov*nfft/100);
    end

    % Process data
    [S, F, T, P] = spectrogram(a.data.y, win.win, nolap, nfft, a.data.fs);

    % Make output AO
    do = xyzdata(T, F, P);
    do.setXunits('s');
```

```
    do.setYunits('Hz');
    do.setZunits(a.data.yunits^2 / unit('Hz'));

    b = ao(do);
    b.setName(sprintf('specgram(%s)', ao_invars{j}), 'internal');
    b.addHistory(getInfo(), pl, cellstr(ao_invars{j}), a.hist);

    % add to output
    bs = [bs b];
  end

  % Reshape the ouput to the same size of the input
  bs = reshape(bs, size(as));

  varargout{1} = bs;
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, 'ao', '', utils.const.categories.sigproc, '$Id: spectrogram.m,v 1.10 2008/09/05 14:15:40 hewitson Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist('Win', getappdata(0, 'ltpda_default_spectral_window'), 'Nolap', -1, 'Nfft', -1);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout=spikecleaning(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  if nargout == 0
    error('### cat cannot be used as a modifier. Please give an output variable.');
  end

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pli             = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  pls = combine(pli, getDefaultPlist());

  % initialise output array
  bo = [];

  % go through each input AO
  for i=1:numel(as)
    a = as(i);
    d = a.data;

    % check this is a time-series object
    if ~isa(d, 'tsdata')
      error(' ### temperature spike detection requires tsdata (time-series) inputs.')
    end

    %--- check input parameters
    kspike = find(pls, 'kspike'); % kspike*sigma definition
    method = find(pls, 'method'); % method of spike-values substitution
    pls.pset('gain', 1);          % gain of the filter
    pls.pset('type', 'highpass'); % type of the filter
```

```
    fs = plist();
    fs.append('fs', d.fs);
    pls.combine(fs);              % determination of the sampling frequency of the input AO

    % high-pass filtering data
    xfiltered = filtfilt(a, miir(pls));

    % standard deviation of the filtered data is calculated
    nxfiltered = find(abs(xfiltered) < kspike*std(xfiltered));

    xfiltered_2 = xfiltered.data.y(nxfiltered);

    std_xfiltered_2 = std(xfiltered_2);

    % spikes vector position is determined
    nspike = find(abs(xfiltered) > kspike*std_xfiltered_2);

    % substitution of spike values starts here
    xcleaned = a.data.y;
    for j=1:length(nspike)
      if nspike(j) <=2 % just in case a spike is detected in the 1st or 2nd sample
        xcleaned(nspike(j)) = mean(xcleaned(1:50));
      else
        if strcmp(method, 'random') % spike is substituted by a random value: N(0,std_xfiltered)
          xcleaned(nspike(j)) = xcleaned(nspike(j)-1) + randn(1)*std_xfiltered_2;
        elseif strcmp(method, 'mean') % spike is substituted by the mean if the two previous values
          xcleaned(nspike(j)) = (xcleaned(nspike(j)-1) + xcleaned(nspike(j)-2))/2;
        elseif strcmp(method, 'previous') % spike is substituted by the pervious value
          xcleaned(nspike(j)) = xcleaned(nspike(j)-1);
        end
      end
    end

    % create new output tsdata
    ts = tsdata(xcleaned, d.fs);
    ts.setYunits(d.yunits);
    ts.setXunits(d.xunits);

    % % create new output history
    % h = history(ALGONAME, VERSION, pls, a.hist);
    % h = set(h, 'invars', invars);

    % make output analysis object
    b = ao(ts);
    b.setName(sprintf('spikecleaning(%s)', ao_invars{i}), 'internal');
    %warning('%%%
```

```
spikecleaning.m,v 1.6 2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%%%
```

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.
%%%
```

```
                    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.
%%%
```

SUBFUNCTIONS ^

- function ii = getInfo(varargin)
- function plo = getDefaultPlist(set)

SOURCE CODE ^

```
% SPLIT split an analysis object into the specified segments.
%%%
```

```
%               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.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = split(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  if nargout == 0
    error('### split cannot be used as a modifier. Please give an output variable.');
  end

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all AOs
  [as, ao_invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
  pli             = utils.helper.collect_objects(varargin(:), 'plist', in_names);

  % copy input plist
  pl = copy(pli, 1);

  %%% go through analysis objects
  bo = [];
  for j=1:numel(as)

    % Unpack parameter list
    split_type = find(pl, 'split_type');

    % check old version without the key 'split_type'
    if ~isempty(find(pl, 'samples'))
      split_type = 'samples';
    elseif ~isempty(find(pl, 'times')) || ...
        ~isempty(find(pl, 'frequencies'))
      split_type = 'times';  % times is the same to frequencies
    end

```

```
    if isempty(split_type)
      error('### please specify the key ''split_type'' in the parameter list');
    end

    % look at input data
    x = as(j).data.getX;
    y = as(j).data.getY;

    %%%%
```

```
      npairs = length(samples);
      if mod(npairs, 2) ~= 0
        error('### please specify a start and stop for each interval.')
      end
      % check data
      if length(x) ~= length(y) && ~isempty(x)
        error('### Something is wrong with the x/y vectors. I can''t split this data.');
      end
      % go over each interval now
      utils.helper.msg(msg.PROC1, sprintf('Split: \f'));
      c = 1;
      dtmp = copy(as(j).data,1);
      dtmp.setXY([],[]);
      for i=1:2:npairs
        utils.helper.msg(msg.PROC1, sprintf('\b%03d  ', round(c)));
        if mod(c,11) == 0 && c < npairs
          utils.helper.msg(msg.PROC1, sprintf('Chunk: \f'));
        end
        is = samples(i);
        ie = samples(i+1);
        % create new output data
        if ~isempty(x)
          x1 = x(is:ie);
        else
          x1 = [];
        end
        y1 = y(is:ie);
        % subtract offset of x for tsdata
        % - we record this info by chaging t0
        d = copy(as(j).data, 1);
        if isa(d, 'tsdata')
          d.setX(x1-x1(1));
        else
          d.setX(x1);
        end
        d.setY(y1);
        % set t0,nsecs if this is tsdata
        if isa(d, 'tsdata')
          t0 = dtmp.t0 + x1(1);
          d.setT0(t0);
          d.setNsecs(length(y1) ./ dtmp.fs);
        end
        % Make AO
        b = ao(d);
        % create new output history
        b.addHistory(getInfo, plist('samples', [is ie]), ao_invars(j), as(j).hist);
        % set name
        b.setName(sprintf('split(%s)[%d]', ao_invars{j},c), 'internal');
        % Add to output array
        bo = [bo b];
        c = c + 1;
      end

      %%%%
```

```
      format_str = find(pl, 'format_str');
      timespan   = find(pl, 'timespan');

      %%% Some checks
      if (~isempty(start_time) || ~isempty(end_time)) && ~isempty(timespan)
        error('### Please specify only a timespan and not additionally the start/end time');
      end

      %%% Convert the start_time into a time object
      if ~isempty(start_time) && ~isa(start_time, 'time')
        start_time = time(start_time);
      end

      %%% Convert the end_time into a time object
      if ~isempty(end_time) && ~isa(end_time, 'time')
        end_time = time(end_time);
      end

      %%% Convert the duration
      if ~isempty(end_time) && ~isempty(duration)
        error('### Please specify only a duration or an end time');
      end
      if ~isempty(duration) && ~isa(duration, 'time')
        duration = time(duration);
        end_time = start_time + duration;
        end_time = time(end_time);
      end

      %%% Set start/end time with a timespan object
      if ~isempty(timespan)
        if ~isa(timespan, 'timespan')
          error('### The timespan must be a timespan object')
        end
        if ~isempty(start_time) || ~isempty(end_time)
          error('### Please specify only a timespan OR a start/end time');
        end

        start_time = timespan.startT;
        end_time   = timespan.endT;
      end

      %%% Set the end_time to the same timezone as the start_time
      end_time.setTimezone(start_time.timezone);

      t0_time = as(j).data.t0;
      t1      = as(j).data.getX(1);

      %%% Comput the start/end time
      ts = (start_time.utc_epoch_milli-t0_time.utc_epoch_milli+t1)/1000;
      te = (end_time.utc_epoch_milli-t0_time.utc_epoch_milli+t1)/1000;

      idx = find(x >= ts & x < te);
      %%% create new output data
      d = copy(as(j).data,1);
      d.setXY(x(idx), y(idx));
      % Set nsecs for tsdata
      if isa(d, 'tsdata')
        if ~isempty(d.x)
          d.setNsecs(d.x(end) - d.x(1) + 1/d.fs);
        else
          d.setNsecs(0);
        end
      end
      % Make AO
      b = ao(d);
      % create new output history
      b.addHistory(getInfo, pl, ao_invars(j), as(j).hist);
      % set name
      b.name = sprintf('split(%s)', ao_invars{j});
      % Add to output array
      bo = [bo b];

    end
  end % for j = 1:numel(as)

  % Set output
  varargout{1} = bo;
end

%--------------------------------------------------------------------------
% Get Info Object
%--------------------------------------------------------------------------

function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
```

```
    sets = {};
    pls   = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {...
      'Default', ...
      'By times', ...
      'By frequencies', ...
      'By samples', ...
      'By chunks', ...
      'By interval start/end', ...
      'By interval start/duration', ...
      'By interval timespan'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ao', '', utils.const.categories.sigproc, '$Id: split.m,v 1.56 2008/09/05 11:05:29 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = plist('split_type', 'times', 'times', []);
    case 'By times'
      plo = plist('split_type', 'times', 'times', []);
    case 'By frequencies'
      plo = plist('split_type', 'frequencies', 'frequencies', []);
    case 'By samples'
      plo = plist('split_type', 'samples', 'samples', []);
    case 'By chunks'
      plo = plist('split_type', 'chunks', 'N', []);
    case 'By interval start/end'
      plo = plist('split_type', 'interval', 'start_time', time(0), 'end_time', time(0));
    case 'By interval start/duration'
      plo = plist('split_type', 'interval', 'start_time', time(0), 'duration', time(0));
    case 'By interval timespan'
      plo = plist('split_type', 'interval', 'timespan', timespan(0,0));
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
% TFE makes transfer function estimates of the time-series objects.
%%%
```

```
function pl = getDefaultPlist()
  pl = plist('Nfft', -1, ...
    'Win', getappdata(0, 'ltpda_default_spectral_window'), ...
    'Olap', -1, ...
    'Order', 0);
end

% END

```

*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 %%%%
```

```
%     bs(kk).data.setY(some_new_data);
%     bs(kk).data.setFs(new_fs);
%     bs(kk).data.setXunits(new_xunits);
%     bd(kk).setName('my name');

    %%% Set Name
%     bs(kk).setName('new name', 'internal');

    %%% Add History
%     bs(kk).addHistory(getInfo('None'), pl, ao_invars(kk), bs(kk).hist);



  %%% Reshape the output to the same size of the input
  varargout{1} = coef;
  varargout{2} = stdx;
  varargout{3} = mse;
  varargout{4} = S;
end

%%%
```

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 ^

```
% XCORR makes cross-correlation estimates of the time-series
%%%
```

```
% Collect input variable names
in_names = cell(size(varargin));
for ii = 1:nargin,in_names{ii} = inputname(ii);end

% Collect all AOs and plists
[as, invars] = utils.helper.collect_objects(varargin(:), 'ao', in_names);
pl               = utils.helper.collect_objects(varargin(:), 'plist', in_names);

% combine plists
pl = combine(pl, getDefaultPlist());

na = numel(as);
if na < 2
  error('### XCORR needs at least two AOs to cross-correlate.');
end

%---------------- Resample all AOs
copies = zeros(size(as));

fsmax = findFsMax(as);
fspl  = plist(param('fsout', fsmax));
for i=1:na
  % check this is a time-series object
  if ~isa(as(i).data, 'tsdata')
    error('### ltpda_xspec requires tsdata (time-series) inputs.');
  end
  % Check Fs
  if as(i).data.fs ~= fsmax
    utils.helper.msg(msg.PROC1, 'resampling AO %s to %f Hz', as(i).name, fsmax);
    % Make a deep copy so we don't
    % affect the original input data
    as(i) = copy(as(i), 1);
    copies(i) = 1;
    as(i).resample(fspl);
  end
end

%---------------- Truncate all vectors

% Get shortest vector
disp('*** Truncating all vectors...');
lmin = findShortestVector(as);
nsecs = lmin / fsmax;
for i=1:na
  if len(as(i)) ~= lmin
    utils.helper.msg(msg.PROC2, 'truncating AO %s to %d secs', as(i).name, nsecs);
    % do we already have a copy?
    if ~copies(i)
      % Make a deep copy so we don't
      % affect the original input data
      as(i) = copy(as(i), 1);
      copies(i) = 1;
    end
    as(i).select(1:lmin);
  end
end

%---------------- check input parameters

% Maximum lag for Xcorr
MaxLag = find(pl, 'MaxLag');

% Scale for Xcorr
scale = find(pl, 'Scale');

% Loop over input AOs
bs(na,na) = ao;
for j=1:na
  for k=1:na
    % -------- Make Xspec estimate

    % Compute cross-correlation estimates using XCORR
    if MaxLag == -1
      MaxLag = len(as(j));
    end
    [c,lags] = xcorr(as(j).data.getY, as(k).data.getY, MaxLag, scale);

    % Keep the data shape of the first input AO
    if size(as(1).data.y,1) == 1
      c = c.';
    end

    % create new output fsdata
    xy = xydata(lags./fsmax, c);
    xy.setXunits('s');
    xy.setYunits('');
```

```
      %----------- create new output history

      % we need to get the input histories in the same order as the inputs
      % to this function call, not in the order of the input to tfestimate;
      % otherwise the resulting matrix on a 'create from history' will be
      % mirrored.
      if j<k
        bs(j,k).addHistory(getInfo('None'), pl, [invars(j) invars(k)], [as(j).hist, as(k).hist]);
      else
        bs(j,k).addHistory(getInfo('None'), pl, [invars(j) invars(k)], [as(k).hist, as(j).hist]);
      end

      % make output analysis object
      bs(j,k).data = xy;
      % set name
      bs(j,k).setName(sprintf('xcorr(%s->%s)', invars{j}, invars{k}), 'internal');
    end % End second loop over AOs
  end % End first loop over AOs

  % Set outputs
  varargout{1} = bs;
end

%-------------------------------------------------------------------------
% Returns the length of the shortest vector in samples
function lmin = findShortestVector(as)
  lmin = 1e20;
  for j=1:length(as)
    if len(as(j)) < lmin
      lmin = len(as(j));
    end
  end
end
%-------------------------------------------------------------------------
% Returns the max Fs of a set of AOs
function fs = findFsMax(as)
  fs = 0;
  for j=1:length(as)
    a = as(j);
    if a.data.fs > fs
      fs = a.data.fs;
    end
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
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, 'ao', '', utils.const.categories.sigproc, '$Id: xcorr.m,v 1.5 2008/09/05 14:15:40 hewitson Exp $', sets, pl);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function plo = getDefaultPlist()

  plo = plist('MaxLag', -1, 'Scale', 'none');

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

acos

PURPOSE ^

ACOS overloads the acos method for Analysis objects.

SYNOPSIS ^

function varargout = acos (varargin)

DESCRIPTION ^

```
 ACOS overloads the acos method for Analysis objects.
%%%
```

```
%               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
%
%%%
```

asin

PURPOSE ^

ASIN overloads the asin method for Analysis objects.

SYNOPSIS ^

function varargout = asin (varargin)

DESCRIPTION ^

```
 ASIN overloads the asin method for Analysis objects.
%%%
```

```
%               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
%
%%%
```

atan

PURPOSE ^

ATAN overloads the atan method for Analysis objects.

SYNOPSIS ^

function varargout = atan (varargin)

DESCRIPTION ^

```
 ATAN overloads the atan method for Analysis objects.
%%%
```

```
%               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
%
%%%
```

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 AO is the same as the data-object
              of the first AO 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 ^

```
% 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 AO is the same as the data-object
```

```
%               of the first AO 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
%
%%%
```

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 ^

```
% 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.
%
```

```
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 %%%%
```

sin

PURPOSE ^

SIN overloads the sin method for Analysis objects.

SYNOPSIS ^

function varargout = sin (varargin)

DESCRIPTION ^

```
 SIN overloads the sin method for Analysis objects.
%%%
```

```
%
% 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
%
%%%
```

```
0094 end
0095
0096
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %
```

```
% 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
%
%%%
```

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(thepzmodel);


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 ));


%%%
```

- ssm2ss SSM2SS converts a statespace model object to a MATLAB statespace object.
- ssmFromBuiltinSystem %%%%

SOURCE CODE ^

```
% 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(thepzmodel);
%
%
% 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 ));
%
%
%%%
```

```
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
```

```
          if isa(varargin{1}, 'ssm') && isa(varargin{2}, 'plist') && isempty(varargin{2}.params)
            % pass to copy constructor
            s = ssm(varargin{1});
          else
            error('### Unknown two parameter constructor.');
          end
        otherwise
          [ss, invars, rest] = utils.helper.collect_objects(varargin(:), 'ssm');
          %%% Do we have a list of SSMs as input
          if ~isempty(ss) && isempty(rest)
            s = ssm(ss);
          else
            error('### Unknown number of constructor arguments.');
          end
      end
    end
    %% copying one ssm
    varargout = copy(varargin)
    %% makes an executable string
    varargout = string(varargin)
    %% change timestep
    varargout = modiftimestep(varargin)
    %% change and subsitute parameters
    varargout = modifparams(varargin)
    %% assemble systems
    varargout = assemble(varargin)
    %% transform into a pzmodel or a iir
    varargout = ssm2iirpz(varargin)
    %% transform into ABCD doubles/symb
    varargout = double(varargin)
    %% transform into a matlab ss object
    varargout = ssm2ss(varargin)
    %% model modification executing a string
    varargout = modify(varargin)
    %% simulation
    varargout = simulate(varargin)
    %% kalman filter
    varargout = kalman(varargin)
    %% model simplification (states)
    varargout = reduce(varargin)
    %% model simplification (variables)
    varargout = reduce_model(varargin)
    %% char
    varargout = char(varargin)
    %% display
    varargout = display(varargin)
    %% give minimal realization (right now needs control toolbox)
    varargout = minreal(varargin)
    %% tells if ssm is stable
    varargout = isstable(varargin)
  end

  %-------- Declaration of private methods --------
  methods (Access = private)
    %% completion and error check
    varargout = validate(varargin)
  end


  %-------- Declaration of Public Static methods --------
  methods (Static=true, Access=public)
    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'ssm');
    end

    function out = VEROUT()
      out = '$Id: ssm.m,v 1.67 2008/09/07 18:16:13 hewitson Exp $';
    end

```

```
    function out = SETS()
      out = {...
        'Default', ...
        'From XML File', ...
        'From MAT File', ...
        'From Built-in Model', ...
        'From Description', ...
        'From Plist', ...
        'From Repository',...
        'From Pzmodel',...
        'From Miir'};
    end

    function pl = getDefaultPlist(set)
      sets = ssm.SETS;
      % Select parameter set
      switch set
        case sets{1} % Default
          pl = plist();
        case sets{2} % XML file
          pl = plist('filename', '');
        case sets{3} % MAT file
          pl = plist('filename', '');
        case sets{4} % built-in
          pl = plist('Built-in', '', 'dim', 3, 'withParams', true);
        case sets{5} % description
          plist('isADescription', '' ,'name','me','ssnames',{'my state'} ,...
            'outputnames',{'force'}, 'inputnames',{'air','food'} ,'timestep',{0}, ...
            'amats',{-0.001}, 'bmats', {0 ,1},'cmats',{1}, 'dmats',{1 , 0}, ...
            'paramnames',{'health'}, 'paramvalues',1,...
            ... %second part, here facultative fields:
            'mmats', {80}, 'amats_handles',cell(1,1), 'bmats_handles',cell(1,2),...
            'cmats_handles',cell(1,1), 'dmats_handles',cell(1,2),...
            'inputvarnames',{'02','hamburger'}, ...
            'ssvarnames', {'energy left'}, ...
            'outputvarnames', {'force left to click on the mouse'},...
            'paramsigmas', 1);
          pl = plist();
        case sets{6} % plist
          pl = plist('plist', plist());
        case sets{7} % repository
          pl = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
        case sets{8} % pzmodel
          pl = plist();
        case sets{9} % miir
          pl = plist();
        otherwise
          error('### Unknown parameter set requested.');
      end
    end

    %% other usefull subroutines
    varargout = cellstrfind(c,s,option)
    varargout = update_struct(varargin)
    %% subroutines for block defined matrix calculus
    a_out = cell_recut(a, rowsizes, colsizes)
    a_out = cell_fusion(a, rowsizes, colsizes)
    c = cell_mult(a,b)
    a = cell_add(a,b)
    [cell_mat_out, rowsizes_out, colsizes_out]  = cell_select(varargin)
  end % End public static methods


  %-------- Declaration of Private Static methods --------
  methods (Static=true, Access=private)
    %% ssm constructor + predefined model reduction
    varargout = ssm_reduced_builtIn(varargin)
    % retrieve from repository
    sys = ssmFromRepository( pl)
```

```
    % load from file
    sys = ssmFromFilename( pl)
    % create from built-in system
    sys = ssmFromBuiltinSystem( pli)
    % create from miir
    sys = ssmFromMiir(miirsin)
    % create from plist
    sys = ssmFromDescription(pl)
    % create from pzmodel
    sys = ssmFromPzmodel(pzmodelsin)
  end

end % End classdef


```

*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 subsytems, 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

validate
%%%

char

PURPOSE ^

CHAR convert a ssm object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a ssm object into a string.
%%%
```

function plo = getDefaultPlist()

SOURCE CODE ^

```
% 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'?)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = char(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  objs = utils.helper.collect_objects(varargin(:), 'ssm');

  pstr = '';

  for ii = 1:numel(objs)
    obj = objs(ii);
    pstr = [pstr sprintf('ssm(name=%s), ', obj.name)];
  end

  pstr = strrep(pstr, '_', '\_');
  pstr = pstr(1:end-2);

  varargout{1} = pstr;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                Local Functions                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getInfo
%
% DESCRIPTION: Get Info Object
%
% HISTORY:     110707 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, 'ssm', '', 'Output', '$Id: char.m,v 1.9 2008/07/21 13:57:30 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     110707 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

display

PURPOSE ^

DISPLAY display ssm object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY display ssm object.
%%%
```

SOURCE CODE ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = display(varargin)
utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);
%%% Check if this is a call for parameters
if utils.helper.isinfocall(varargin{:})
  varargout{1} = getInfo(varargin{3});
  return
end
objs = utils.helper.collect_objects(varargin(:), 'ssm');
txt = utils.helper.objdisp(objs);
if nargout == 0
  for ii=1:length(txt)
    disp(txt{ii});
  end
else
  varargout{1} = txt;
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                   Local Functions                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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, 'ssm', '', 'Output', '$Id: display.m,v 1.1 2008/08/06 18:35:26 ingo Exp $', sets, pl);
end

function plo = getDefaultPlist()
  plo = plist();
end
```

isstable

PURPOSE ^

tells if ssm is numerically stable

SYNOPSIS ^

function varargout = isstable(varargin)

DESCRIPTION ^

```
 tells if ssm is numerically stable
%%%
```

```
%                   Creation
%
%%%
```

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.
%%%
```

```
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

assemble

PURPOSE ^

assembles embedded subsytems, with exogenous inputs

SYNOPSIS ^

function varargout = assemble(varargin)

DESCRIPTION ^

```
 assembles embedded subsytems, with exogenous inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 DESCRIPTION: assemble assembles embedded subsytems, 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

%%%

```
% Collect all SSMs and plists
[sys, ssm_invars] = utils.helper.collect_objects(varargin(:), 'ssm', in_names);

Nsys = numel(sys);

if Nsys < 2
  error('### Two or more systems are needed to assemble.');
end

% We want to force a copy - this is not a modify method
if nargout < 1
  error('### assemble cannot be used as a modifier. Please give an output variable.');
end

% Decide on a deep copy or a modify, depending on the output
sys_array = copy(sys, nargout);

%% begin function body
sys_out = ssm;

%%  Putting toghether parameter fields
for i=1:Nsys
  sys_out.paramnames  = [sys_out.paramnames  sys_array(i).paramnames ];  %#ok<AGROW>
  sys_out.paramvalues = [sys_out.paramvalues sys_array(i).paramvalues];  %#ok<AGROW>
  sys_out.paramsigmas = [sys_out.paramsigmas sys_array(i).paramsigmas];  %#ok<AGROW>
end
sys_out.isnumerical = true;
for i=1:Nsys
  sys_out.isnumerical = sys_out.isnumerical*sys_array(i).isnumerical;
end

%% merging ss data : ss* output* *input

names       = cell(1,0);
ssnames       = cell(1,0);
ssvarnames    = cell(1,0);
sssizes       = zeros(1,0);
ssposition    = zeros(2,0);
outputnames    = cell(1,0);
outputvarnames = cell(1,0);
outputsizes    = zeros(1,0);
outputposition = zeros(2,0);
inputnames    = cell(1,0);
inputvarnames = cell(1,0);
inputsizes    = zeros(1,0);
inputposition = zeros(2,0);
for i=1:Nsys
  names           = [names           sys_array(i).name           ];
  ssnames         = [ssnames         sys_array(i).ssnames        ];
  ssvarnames      = [ssvarnames      sys_array(i).ssvarnames     ];
  sssizes         = [sssizes         sys_array(i).sssizes        ];
  ssposition      = [ssposition      [i*ones(1,sys_array(i).Nss)      ; 1:sys_array(i).Nss]       ];
  outputnames     = [outputnames     sys_array(i).outputnames    ];
  outputvarnames  = [outputvarnames  sys_array(i).outputvarnames ];
  outputsizes     = [outputsizes     sys_array(i).outputsizes    ];
  outputposition  = [outputposition  [i*ones(1,sys_array(i).Noutputs) ; 1:sys_array(i).Noutputs]  ];
  inputvarnames   = [inputvarnames   sys_array(i).inputvarnames  ];
  inputnames      = [inputnames      sys_array(i).inputnames     ];
  inputsizes      = [inputsizes      sys_array(i).inputsizes     ];
  inputposition   = [inputposition   [i*ones(1,sys_array(i).Ninputs)  ; 1:sys_array(i).Ninputs ]  ];
end
%% data already good to store
sys_out.ssnames            = ssnames;
sys_out.ssvarnames         = ssvarnames;
sys_out.outputnames        = outputnames;
sys_out.outputvarnames     = outputvarnames;

%% non redundancy checks
for i=1:length(names)
```

```
    [pos_in, sum_in] = ssm.cellstrfind(names, names{i},'all');
    if sum_in>1
      error([num2str(sum_in),' systems are named ', names{i}])
    end
  end
  for i=1:length(ssnames)
    [pos_in, sum_in] = ssm.cellstrfind(ssnames, ssnames{i},'all');
    if sum_in>1
      error([num2str(sum_in),' states are named ', ssnames{i}])
    end
  end
  for i=1:length(outputnames)
    [pos_in, sum_in] = ssm.cellstrfind(outputnames, outputnames{i},'all');
    if sum_in>1
      error([num2str(sum_in),' outputs are named ', outputnames{i}])
    end
  end

  %% fields for outer inputs
  inputisouter = false(1,length(outputnames));
  for i=1:length(inputnames)
    [pos_in, sum_in] = ssm.cellstrfind(outputnames, inputnames(i),'all');
    if sum_in==0
      inputisouter(i)=true;
    end
  end

  %% building A, M and C matrices
  amats = {};
  mmats = {};
  cmats = {};
  for i=1:Nsys
    amats = ...
      [amats                        cell(size(amats,1),size(sys_array(i).amats,2))    ; ...
      cell(size(sys_array(i).amats,1),size(amats,2))            sys_array(i).amats ] ;
    mmats = ...
      [mmats                        cell(size(mmats,1),size(sys_array(i).mmats,2))    ; ...
      cell(size(sys_array(i).mmats,1),size(mmats,2))            sys_array(i).mmats ] ;
    cmats = ...
      [cmats                        cell(size(cmats,1),size(sys_array(i).cmats,2))    ; ...
      cell(size(sys_array(i).cmats,1),size(cmats,2))            sys_array(i).cmats ] ;
  end
  sys_out.mmats = mmats;

  %% building B_in and D_in matrices
  B_in = cell( length(ssnames),     length(outputnames) );
  D_in = cell( length(outputnames), length(outputnames) );
  for i=1:length(outputnames)
      [pos_in, sum_in] = ssm.cellstrfind(inputnames, outputnames{i},'all');
      if sum_in>0
        for pos = pos_in
          systemNumber = inputposition(1,pos);
          inputNumber  = inputposition(2,pos);
            B_in(ssposition(1,:)==systemNumber, i)     = sys_array(systemNumber).bmats(:,inputNumber);
            D_in(outputposition(1,:)==systemNumber, i) = sys_array(systemNumber).dmats(:,inputNumber);
        end
      end
  end

  %% building B_ext and D_ext matrices
  B_ext = cell( length(ssnames),     length(inputnames) );
  D_ext = cell( length(outputnames), length(inputnames) );
  pos_keep = false(1, length(inputnames));
  for i=1:length(inputnames)
      [pos_in, sum_in] = ssm.cellstrfind(outputnames, inputnames(i),'all');
      if sum_in==0
        [pos_in, sum_in] = ssm.cellstrfind(inputnames, inputnames(i),'all');
        for pos = pos_in;
          systemNumber  = inputposition(1,pos);
```

```
          inputNumber   = inputposition(2,pos);
          B_ext( ssposition(1,:)==systemNumber , pos_in(1) )    = sys_array(systemNumber).bmats(:,inputNumber);
          D_ext( outputposition(1,:)==systemNumber, pos_in(1) ) = sys_array(systemNumber).dmats(:,inputNumber);
          pos_keep(pos_in(1)) = true;
        end
      end
  end

  %% getting amats, bmats, cmats, dmats
  B_ext                  = B_ext(:,pos_keep);
  D_ext                  = D_ext(:,pos_keep);
  sys_out.inputnames    = inputnames(pos_keep);
  sys_out.inputvarnames = inputvarnames(pos_keep);

  %% flattening and getting constant feedthrough
  D_in_f = ssm.cell_fusion(D_in, outputsizes,outputsizes);
  id_D_inv = (eye(size(D_in_f)) - D_in_f)^-1;
  id_D_inv = ssm.cell_recut(id_D_inv, outputsizes, outputsizes);
  sys_out.amats = ssm.cell_add(amats, ssm.cell_mult(B_in, ssm.cell_mult(id_D_inv, cmats)));
  sys_out.bmats = ssm.cell_add(B_ext, ssm.cell_mult(B_in, ssm.cell_mult(id_D_inv, D_ext)));
  sys_out.cmats = ssm.cell_mult(id_D_inv, cmats);
  sys_out.dmats = ssm.cell_mult(id_D_inv, D_ext);

  %% building new name and strings
  name = 'assembled( ';
  for i = 1:Nsys
    if i==1
      name = [name, sys_array(i).name ];                        %#ok<AGROW>
    elseif i<Nsys
      name = [name,' + ', sys_array(i).name];                   %#ok<AGROW>
    else
      name = [name,' + ', sys_array(i).name,')'];               %#ok<AGROW>
    end
  end
  name = [name ')'];
  sys_out.name = name;

  %% getting timestep and checking consitency
  for i=1:Nsys
    if i == 1
      sys_out.timestep = sys_array(i).timestep;
    elseif ~ sys_out.timestep == sys_array(i).timestep
      error(['error because systems 1 and ',num2str(i),...
        ' named ',sys_array(i).name,' and ',sys_array(i).name,...
        ' have different timesteps :',...
        num2str(sys_array(i).timestep),' and ',num2str(sys_array(i).timestep) ]);
    end
  end

  %% setting history
  h = history();
  for i=1:Nsys
    h(i) = sys_array(i).hist;
  end
  sys_out.addHistory(ssm.getInfo(mfilename), plist , {''}, h );
  %% parsing output
  validate(sys_out);
  varargout = {sys_out};
end


%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
```

```
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function out = getDefaultPlist(set)
  switch set
    case 'Default'
      out = plist;
    otherwise
      error('### Unknown set [%s] to get the default plist.', set);
  end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

- 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 ^

```
% Convert a statespace model object to double arrays for given i/o
%%%
```

```
function varargout = double(varargin)

  %% starting initial checks
  utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % checking number of inputs work data
  if ~nargin==2, error('wrong number of inputs!'), end

  if ~isequal(class(varargin{1}),'ssm'), error(['argument is not a ssm but a ', class(varargin{1})]),end
  if ~isequal(class(varargin{2}),'plist'), error(['argument is not a plist but a ', class(varargin{2})]),end

  sys = varargin{1};
  options = combine(varargin{2}, getDefaultPlist('Default'));
  %% begin function body
  %making model simplifications
  sys = reduce_model(sys, options);

  %  Convert to double arrays

  M =          ssm.cell_fusion(sys.mmats, sys.sssizes,     sys.sssizes);
  A = (M^-1) * ssm.cell_fusion(sys.amats, sys.sssizes,     sys.sssizes);
  B = (M^-1) * ssm.cell_fusion(sys.bmats, sys.sssizes,     sys.inputsizes);
  C =          ssm.cell_fusion(sys.cmats, sys.outputsizes, sys.sssizes);
  D =          ssm.cell_fusion(sys.dmats, sys.outputsizes, sys.inputsizes);
  Ts = sys.timestep;

  inputvarnames3 = cell(1,sum(sys.inputsizes));
  k=1;
  for i=1:sys.Ninputs
    for j=1:sys.inputsizes(i)
      inputvarnames3{k} = sys.inputvarnames{i}{j};
      k = k+1;
    end
  end
  ssvarnames3 = cell(1,sum(sys.sssizes));
  k=1;
  for i=1:sys.Nss
    for j=1:sys.sssizes(i)
      ssvarnames3{k} = sys.ssvarnames{i}{j};
      k = k+1;
    end
  end
  outputvarnames3 = cell(1,sum(sys.outputsizes));
  k=1;
  for i=1:sys.Noutputs
    for j=1:sys.outputsizes(i)
      outputvarnames3{k} = sys.outputvarnames{i}{j};
      k = k+1;
    end
  end


%% parsing output
  varargout = {A B C D Ts inputvarnames3 ssvarnames3 outputvarnames3};

end


function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
end
```

```
function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = ssm.getInfo('reduce_model', 'Default').plists;
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
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
%%%
```

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 ^

```
%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
%%%
```

```
true), options);

  options = plist('inputs', noise_varnames, 'outputs', outputs_varnames, 'states', 'ALL'
);
  [A E C F Ts noise_varnames2 ss_varnames2 outputs_varnames2] = double(copy(sys(i_sys),
true), options);

  options = plist('inputs', constants_varnames , 'outputs', outputs_varnames, 'states',
'ALL'  );
  [A G C H Ts constants_varnames2 ss_varnames2 outputs_varnames2] =
double(copy(sys(i_sys), true), options);

  %% evaluating Kalman feedback K, innovation gain M, state covariance P, output
covariance Z
  % given Q and R (process and measurement noise covariances)
  Qn = E*noise_in*transpose(E);
  Qn = (Qn + 1e-10*norm(Qn)*eye(size(Qn)));
  Rn = F*noise_in*transpose(F);
  Rn = Rn + 1e-10*norm(Rn)*eye(size(Rn));
  Nn = E*noise_in*transpose(F);
  P = eye(size(A))*1e20;
  for i=1:1000
    P = A*P*A'+Qn;
    K = P*C'*(C*P*C'+Rn)^-1;
    P = (eye(size(A)) - K*C)*P;
  end
  Z = C*P*C' + Rn;

  %% dividing or modifying size of matrices
  As = ssm.cell_recut(A, sssizes, sssizes);
  Bs = ssm.cell_recut(B, sssizes, Naos_in);
  Gs = ssm.cell_recut(G, sssizes, Nconstants_in);
  Cs = ssm.cell_recut(C, Noutputs_in, sssizes);
  Ks = ssm.cell_recut(K, sssizes, Noutputs_in);

  %% getting data position in new matrices

  for i=1:Naos_in
      % getting position
      [positions, sum, value] = ssm.cellstrfind(aos_varnames,aos_varnames2{i},'all');
      % checking against redundant input data
      if sum==1
        aos_pos(positions) = i;
        aos_data{positions} = aos_in(positions).data.getY ;
      elseif sum>1
        error(['more than one so variable is named ', value]);
      end
  end
  for i=1:Nconstants_in
      % getting position
      [positions, sum, value] =
ssm.cellstrfind(constants_varnames,constants_varnames2{i},'all');
      % checking against redundant input data
      if sum==1
        constants_pos(positions) = i;
      elseif sum>1
        error(['more than one constant variable is named ', value]);
      end
  end
  for i=1:Noutputs_in
      % getting position
      [positions, sum, value] =
ssm.cellstrfind(outputs_varnames,outputs_varnames2{i},'all');
      % checking against redundant output data
      if sum==1
        outputs_pos(positions) = i;
      elseif sum>1
        error(['more than one output variable is named ', value]);
      end
  end

  %% getting correct number of samples
  Nsamples = inf;
  f0 = 1/timestep;
  for i=1:Naos_in
    Nsamples = min(Nsamples,length(aos_in(i).data.y));
    if ~(f0==aos_in(i).data.fs)
      str = ['WARNING : ssm frequency is ',num2str(f0),...
        ' but sampling frequency of ao named ',...
        aos_in(i).name, ' is ', num2str(aos_in(i).data.fs) ];
      utils.helper.msg(utils.const.msg.MNAME, str);
    end
    % maybe tdata should be retrieved and verified to be equal, rather than this.
  end
  if Nsamples == inf % case there is no input!
    Nsamples = 0;
```

```
end

%% evaluating data to return
return_states = find(plist_inputs, 'return states');
returnStates = false(1,Nss);
for i=1:Nss
  returnStates(i) = ~isempty(find( ismember(return_states, i)));
end
return_outputs = find(plist_inputs, 'return outputs');
returnOutputs = ~isempty(find( ismember(return_outputs, 1)));

%% simulation first step
x = cell(Nss, 1);
lastX = ssini;
for i=1:Nss
  if returnStates(i)
    x{i} = zeros(sssizes(i), Nsamples);
  end
end
if returnOutputs
  y = zeros(Noutputs, Nsamples);
end
IncrementX = cell(Nss,1);

%% simulation loop
for k = 1:Nsamples
  if rem(k,500)==0
    utils.helper.msg(utils.const.msg.MNAME, ['filtering time : ',num2str(k*timestep)
]);
  end

  %% resetting fields
  for s = 1:Nss
    IncrementX{s} = zeros(sssizes(s),1);
  end
  IncrementY = zeros(Noutputs_in,1);

  %% prediction step
  for i=1:Naos_in
    for s=1:Nss
      IncrementX{s} = IncrementX{s} +Bs{s}(:,aos_pos(i))*aos_data{i}(k);
    end
     IncrementY = IncrementY +D(:,aos_pos(i))*aos_data{i}(k);
  end
  for i=1:Nconstants_in
    for s=1:Nss
      IncrementX{s} = IncrementX{s} + Gs{s}(:,constants_pos(i))*constants_in(i);
    end
     IncrementY = IncrementY + H(:,constants_pos(i))*constants_in(i);
  end

  % differential equation
  lastX = ssm.cell_add(IncrementX, ssm.cell_mult(As, lastX));

  % output equations
  lastY = zeros(Noutputs_in,1);
  for s=1:Nss
    lastY = lastY + Cs{s}*lastX{s};
  end
  lastY = lastY + IncrementY;

  thisY = zeros(Noutputs_in,1);
  for i=1:Noutputs_in
    thisY(outputs_pos(i)) = outputs_in(i).data.y(k);
  end

  %% state correction (=update) step
  for i=1:Noutputs_in
    for s=1:Nss
      lastX{s} = lastX{s} + Ks{s}*( thisY - lastY) ;
    end
  end
  % output correction
  lastY = zeros(Noutputs_in,1);
  for s=1:Nss
    lastY = lastY + Cs{s}*lastX{s};
  end
  lastY = lastY + IncrementY;

  %% computing and storing outputs

  for s=1:Nss
    if returnStates(i)
      x{s}(:,k) = lastX{s};
    end
  end
```

```
    if returnOutputs
      y(:,k) = lastY;
    end
  end
  %% saving in aos
  Naos = 0;
  for i=1:Nss
    if returnStates(i)
      Naos = Naos + sssizes(i);
    end
  end
  if returnOutputs
    Naos = Naos + Noutputs_in;
  end
  ao_out(Naos) = ao;
  if ~exist('t','var')
    t = (1:Nsamples)*timestep;
  end
  count = 1;
  for i=1:Nss
    for j=1:sssizes(i)
      if returnStates(i)
        ao_out(count) = ao(tsdata(t,x{i}(j,:)));
        ao_out(count).setName(['kal_X_',num2str(i),'_',num2str(j)]);
        ao_out(count).setDescription(...
          ['ssm#: ',num2str(i_sys),...
          ', state: ',sys(i_sys).ssnames{i},' ',sys(i_sys).ssvarnames{i}{j}]);
        ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs ,
{''}, sys(i_sys).hist );
        count = count+1;
      end
    end
  end
  for i=1:Noutputs_in
    if returnOutputs
      ao_out(count) = ao(tsdata(t,y(i,:)));
      ao_out(count).setName(['kal_Y_',num2str(i)]);
      ao_out(count).setDescription(...
        ['ssm#: ',num2str(i_sys),', output: ',outputs_varnames2{i}] );
      ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs ,
{''}, sys(i_sys).hist );
      count = count+1;
    end
  end
  %% construct output analysis object
  plist_data_out = plist('K', K, 'P', P, 'Z', Z);
  varargout  = {ao_out, plist_data_out};
end

function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls   = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38
ingo Exp $', sets, pls);
end

function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = plist(...
        'noise variable names',   {'noise_readout', 'noise_special'}, ...
        'variance',               [1 0.1;1 0.1],...
        'aos variable names',     {'input_1' 'input_2'},...
        'aos',                    [ao, ao],...
        'constant variable names',{ 'noise_readout' 'noise_readout2'}, ...
        'constant',               [1;6], ...
        'output variable names',  {'output_1'}, ...
        'outputs',                ao , ...
        'return states',          [], ...
        'return outputs',         [1] , ...
        'ssini',                  {} ...
        );
    otherwise
      error('### Unknown parameter set [%s].', set);
```

```
0387    end
0388 end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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

%%%

```
if ~nargin==2, error('wrong number of inputs!'), end

if ~isequal(class(varargin{1}),'ssm'), error(['argument is not a ssm but a ', class(varargin{1})]),end

sys = varargin{1};

%% begin function body
for i_sys=1:length(sys)
  if ~sys(i_sys).isnumerical
    error('system should be numeric')
  end
  ss = ssm2ss(sys(i_sys), plist);
  ss = minreal(ss);
  [A,B,C] = ssdata(ss);
  sys(i_sys).amats = {A};
  sys(i_sys).mmats = {eye(size(A))};

  bmats = cell(1, sys(i_sys).Ninputs);
  inputranks = cumsum([1, sys(i_sys).inputsizes]);
  for i=sys(i_sys).Ninputs
    bmats{i} = B(:, inputranks(i):(inputranks(i+1)-1));
  end

  cmats = cell(sys(i_sys).Noutputs,1);
  outputranks = cumsum([1, sys(i_sys).outputsizes]);
  for i=sys(i_sys).Noutputs
    cmats{i} = C(outputranks(i):(outputranks(i+1)-1), :);
  end

  sys(i_sys).bmats = bmats;
  sys(i_sys).cmats = cmats;
  sys(i_sys).amats_handles = cell(1,1);
  sys(i_sys).bmats_handles = cell(1,sys(i_sys).Ninputs);
  sys(i_sys).cmats_handles = cell(sys(i_sys).Noutputs,1);
  sys(i_sys).dmats_handles = {};
  sys(i_sys).sssizes =[];
  sys(i_sys).ssnames = sys(i_sys).ssnames(1);
  sys(i_sys).ssvarnames = {};

  if (sys(i_sys).Nss > 1)
    utils.helper.msg(utils.const.msg.MNAME, ['warning in ', mfilename, ' : state space blocks will be fusionned!!!']);
  end
  sys(i_sys) = validate(sys(i_sys));
end
varargout{1} = sys;

end


function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
end

function plo = getDefaultPlist(set)
  switch set
```

```
    case 'Default'
      plo = plist();
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
end

```

*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
%%%
```

```
 Creation

 TODO : Modify for multiple outputs
 detect case when parameter is Set twice in a list
 inplement error message if keep is not empty and no symbollic output is
 allowed
%%%
```

```
%           (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
% inplement error message if keep is not empty and no symbollic output is
% allowed
%%%
```

```
      if length(i_options)>1
        sys(i_sys, i_options) = sys(i_sys);
      end

      %% building updated list for KeepParam...
      if isequal(KeepParamOrder,'NONE') || SymbolicOutputAllowed==0
        KeepParamNames = {};
      elseif isequal(KeepParamOrder,'LIST')
        KeepParamNames = KeepParamNamesIni;
      elseif isequal(KeepParamOrder,'ALL')
        KeepParamNames = sys(i_sys).paramnames;
      else
        KeepParamNames = {};
        j=1;
        for i=1:length(KeepParamNamesIni)
          if KeepParamOrder(i)
            KeepParamNames{j} = KeepParamNamesIni{i}; %#ok<AGROW>
            j = j+1;
          end
        end
      end
      nKeep = length(KeepParamNames);

      %% building updated list for SetParamValues
      if (isequal(SetParamValuesOrder,'LIST'))
        SetParamValuesNames = SetParamNamesIni;
        SetParamValuesOrder = ones(length(SetParamNamesIni),1);
        SetParamValues = SetParamValuesIni;
      elseif (isequal(SetParamValuesOrder,'NONE'))
        SetParamValuesNames = {};
        SetParamValuesOrder = [];
        SetParamValues = [];
      else
        SetParamValuesNames = SetParamNamesIni;
        SetParamValues = SetParamValuesIni;
      end
      nValues = length(SetParamValues);

      %% building updated list for SetParamSigmas
      if (isequal(SetParamSigmasOrder,'LIST'))
        SetParamSigmasNames = SetParamNamesIni;
        SetParamSigmasOrder = ones(length(SetParamNamesIni),1);
        SetParamSigmas = SetParamSigmasIni;
      elseif (isequal(SetParamSigmasOrder,'NONE'))
        SetParamSigmasNames = {};
        SetParamSigmasOrder = [];
        SetParamSigmas = [];
      else
        SetParamSigmasNames = SetParamNamesIni;
        SetParamSigmas = SetParamSigmasIni;
      end
      nSigmas = length(SetParamSigmas);

      %% building param position tables
      posValues = zeros(nValues,1);
      posKeep = zeros(nKeep,1);
      posSigmas = zeros(nSigmas,1);
      for i=1:sys(i_sys).Nparams
        for j=1:nKeep
          if isequal(sys(i_sys).paramnames{i},KeepParamNames{j})
            posKeep(j) = i;
          end
        end
        for j=1:nValues
          if isequal(sys(i_sys).paramnames{i},SetParamValuesNames{j})
            posValues(j) = i;
          end
        end
        for j=1:nSigmas
```

```
            if isequal(sys(i_sys).paramnames{i},SetParamSigmasNames{j})
              posSigmas(j) = i;
            end
          end
        end

        %% looking for non exisiting but modified parameter names
        if not(nKeep==sum(posKeep>0))
          msg = 'error with keep parameter list because one does not exist in system.';
          error(msg);
        elseif not(nValues==sum(posValues>0))
          msg = 'error with SetValues parameter list because one does not exist in system.';
          error(msg);
        elseif not(nSigmas==sum(posSigmas>0))
          msg = 'error with SetSigma list because one does not exist in system.';
          error(msg);
        end

        %% modifying parameter tables in sys(i_sys) (Setparam list)
        for i=1:nValues
          if SetParamValuesOrder(i)
            sys(i_sys).paramvalues(posValues(i)) = SetParamValues(i);
          end
        end
        for i=1:nSigmas
          if SetParamSigmasOrder(i)
            sys(i_sys).paramsigmas(posSigmas(i)) = SetParamSigmas(i);
          end
        end

        %% Numerical value for symbolic variables
        for i=1:sys(i_sys).Nparams
          if sum(ismember(posKeep, i))==0
            cmd1 = ['sym(''', sys(i_sys).paramnames{i}, ''');'];
            cmd2 = [sys(i_sys).paramnames{i}, '= sys(i_sys).paramvalues(', num2str(i), ');'];
            eval(cmd1);
            eval(cmd2);
          else
            cmd1 = [sys(i_sys).paramnames{i}, '=sym(''', sys(i_sys).paramnames{i}, ''');'];
            eval(cmd1);
          end
        end

        %% keeping only params in 'keep param names'
        sys(i_sys,i_options).paramnames  = sys(i_sys,i_options).paramnames(posKeep);
        sys(i_sys,i_options).paramvalues = sys(i_sys,i_options).paramvalues(posKeep);
        sys(i_sys,i_options).paramsigmas = sys(i_sys,i_options).paramsigmas(posKeep);

        %% evaluation of A, B, C, D matrices
        for i_ss=1:sys(i_sys,i_options).Nss
          for j_ss =1:sys(i_sys,i_options).Nss
            %Mass Matrix
            if ~isempty(sys(i_sys,i_options).mmats{i_ss, j_ss})
              if ~isnumeric(sys(i_sys,i_options).mmats{i_ss, j_ss})
                sys(i_sys,i_options).mmats{i_ss, j_ss} = ...
                  eval(sym(sys(i_sys,i_options).mmats{i_ss, j_ss}));
              end
            end
            %A Matrix
            if ~isempty(sys(i_sys,i_options).amats{i_ss, j_ss})
              if ~isnumeric(sys(i_sys,i_options).amats{i_ss, j_ss})
                sys(i_sys,i_options).amats{i_ss, j_ss} = ...
                  eval(sym(sys(i_sys,i_options).amats{i_ss, j_ss}));
              end
            end
          end
        end
        for i_ss=1:sys(i_sys,i_options).Nss
          for j_in =1:sys(i_sys,i_options).Ninputs
```

```
          %B Matrix
          if ~isempty(sys(i_sys,i_options).bmats{i_ss, j_in})
            if ~isnumeric(sys(i_sys,i_options).bmats{i_ss, j_in})
              sys(i_sys,i_options).bmats{i_ss, j_in} = ...
                eval(sym(sys(i_sys,i_options).bmats{i_ss, j_in}));
            end
          end
        end
      end
      for i_out=1:sys(i_sys,i_options).Noutputs
        for j_ss =1:sys(i_sys,i_options).Nss
          %C Matrix
          if ~isempty(sys(i_sys,i_options).cmats{i_out, j_ss})
            if ~isnumeric(sys(i_sys,i_options).cmats{i_out, j_ss})
              sys(i_sys,i_options).cmats{i_out, j_ss} = ...
                eval(sym(sys(i_sys,i_options).cmats{i_out, j_ss}));
            end
          end
        end
      end
      for i_out=1:sys(i_sys,i_options).Noutputs
        for j_in =1:sys(i_sys,i_options).Ninputs
          %D Matrix
          if ~isempty(sys(i_sys,i_options).dmats{i_out, j_in})
            if ~isnumeric(sys(i_sys,i_options).dmats{i_out, j_in})
              sys(i_sys,i_options).dmats{i_out, j_in} = ...
                eval(sym(sys(i_sys,i_options).dmats{i_out, j_in}));
            end
          end
        end
      end
      validate(sys(i_sys,i_options));
      sys(i_sys,i_options).addHistory(ssm.getInfo(mfilename), options(i_options) , {''}, sys(i_sys,i_options).hist );
    end
  end
  if nargout > 0
    varargout{1} = sys;
  end
end


%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls   = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
end

%-------------------------------------------------------------------------
% Get Default Plist
%-------------------------------------------------------------------------
function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = plist( 'KEEP PARAM NAMES', {}, 'KEEP PARAM ORDER', 'LIST',...
```

```
        'SET_PARAM_NAMES', {}, 'SET_PARAM_VALUES', [], 'SET_PARAM_SIGMAS', [],...
        'SET_PARAM_VALUES_ORDER', 'LIST', 'SET_PARAM_SIGMAS_ORDER', 'NONE', 'SYMBOLIC_OUTPUT_ALLOWED', 1);
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

modiftimestep

PURPOSE ^

modiftime modifies the timestep of a ssm object

SYNOPSIS ^

function varargout = modiftimestep(varargin)

DESCRIPTION ^

```
 modiftime modifies the timestep of a ssm object
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 DESCRIPTION: modiftime 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('modiftimestep')

              Get information about a specified set-plist by calling:
              >> ssm.getInfo('modiftimestep', '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 ^

```
% modiftime modifies the timestep of a ssm object
%%%
```

```
      mmat       = ssm.cell_fusion(sys_in(i).mmats, sssizes, sssizes);
      bmat       = ssm.cell_fusion(sys_in(i).bmats, sssizes, inputsizes);
      %% getting rid of M matrix
      minv = mmat^(-1);
      %      minv = double(vpa(mmat,30)^(-1));
      amat = minv*amat;
      bmat = minv*bmat;
      mmat_2 = eye(size(amat));
      %% take different actions depending on old and new timestep
      if timestep == newtimestep
        action = 'DoNothing';
      elseif newtimestep == 0
        action = 'MakeContinuous';
        str = 'warning because system is sent back to continuous time';
        utils.helper.msg(utils.const.msg.MNAME, str);
      elseif timestep == 0
        action = 'Discretize';
      elseif floor(newtimestep/timestep) == newtimestep/timestep
        action = 'TakeMultiple';
      elseif newtimestep > timestep
        action = 'MakeLonger';
      else
        action = 'MakeShorter';
        str = 'warning because system is sent back to shorter time step';
        utils.helper.msg(utils.const.msg.MNAME, str);
      end

      %% proceed with matrix modifications
      NSubstep = 35;
      amat_input = zeros(size(amat));
      if isequal(action,'DoNothing')
        %% same timestep : do nothing
        amat_2  = amat;
        bmat_2 = bmat;
      elseif isequal(action,'Discretize')
        %% from continuous to discrete
        amat_2 = expm(amat * newtimestep);
        for i_step = 1:NSubstep
          position = newtimestep*(2*i_step-1)/(2*NSubstep);
          amat_input = amat_input + expm(amat*position);
        end
        amat_input = amat_input*newtimestep/NSubstep;
        bmat_2 = real(amat_input * bmat);
      elseif isequal(action,'MakeContinuous')
        %% for discrete to continuous
        [V, E] = eig(amat);
        amat_2 = V * (diag(diag(log(E)))/timestep) * V^(-1);
        for i_step = 1:NSubstep
          position = timestep*(2*i_step-1)/(2*NSubstep);
          amat_input = amat_input + expm( amat_2*position );
        end
        amat_input = amat_input*timestep/NSubstep;
        amat_input_inv = (amat_input)^(-1);
        bmat_2 = real(amat_input_inv * bmat);
      elseif isequal(action,'TakeMultiple')
        %% discrete to discrete with a multiple
        multiple = newtimestep/timestep;
        amat_2 = amat^multiple;
        for i_step = 1:multiple
          amat_input = amat_input + amat^(i_step-1);
        end
        bmat_2 = real(amat_input * bmat);

      elseif isequal(action,'MakeLonger')||isequal(action,'MakeShorter')
        %% discrete to discrete with no multiple relationship
        [V, E] = eig(amat);
        amat_1 = V * (diag(diag(log(E)))/timestep) * V^(-1);
        amat_2 = expm( amat_1 * newtimestep);
        amat_input_1 = amat_input;
        amat_input_2 = amat_input;
        for i_step = 1:NSubstep
          position_1 = timestep*(2*i_step-1)/(2*NSubstep);
          position_2 = newtimestep*(2*i_step-1)/(2*NSubstep);
          amat_input_1 = amat_input_1 + expm( amat_1*position_1 );
          amat_input_2 = amat_input_2 + expm( amat_1*position_2 );
        end
        amat_input_1 = amat_input_1*timestep/NSubstep;
        amat_input_2 = amat_input_2*newtimestep/NSubstep;
        amat_input_1inv = (amat_input_1)^(-1);
        bmat_2 = real(amat_input_2 * amat_input_1inv * bmat);
      end
      amat_2 = real(amat_2);
      bmat_2 = real(bmat_2);
      %% Save Matrix modifications
      sys_out(i,j).amats = ssm.cell_recut(amat_2, sssizes, sssizes);
      sys_out(i,j).mmats = ssm.cell_recut(mmat_2, sssizes, sssizes);
```

```
      sys_out(i,j).bmats = ssm.cell_recut(bmat_2, sssizes, inputsizes);
      sys_out(i,j).timestep = newtimestep;
      sys_out(i,j).addHistory(ssm.getInfo(mfilename), options(j) , {''}, sys_in(i).hist
);
    end
  end
  if ~(Noptions==1)
    varargout = {sys_out};
  else
    varargout = {sys_in};
  end
end


function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls    = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38
ingo Exp $', sets, pls);
end

function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = plist('newtimestep',0.1);
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = modify(varargin)

  %% starting initial checks
  utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % Collect input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin,in_names{ii} = inputname(ii);end

  % Collect all SSMs and plists
  [sys, ssm_invars] = utils.helper.collect_objects(varargin(:), 'ssm', in_names);
  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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

reduce

PURPOSE ^

reduce enables to do model simplification

SYNOPSIS ^

function varargout = reduce(varargin)

DESCRIPTION ^

```
 reduce enables to do model simplification
%%%
```

- ssm SSM statespace model class constructor.

SUBFUNCTIONS ^

- function ii = getInfo(varargin)
- function plo = getDefaultPlist(set)

SOURCE CODE ^

```
% 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 :
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = reduce(varargin)

  %% starting initial checks
  utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % checking number of inputs work data
  if ~nargin==2, error('wrong number of inputs!'), end

  if ~isequal(class(varargin{1}),'ssm'), error(['argument is not a ssm but a ', class(varargin{1})]),end
  if ~isequal(class(varargin{2}),'plist'), error(['argument is not a plist but a ', class(varargin{2})]),end

  sys = varargin{1};
  options = varargin{2};
  Nsys     = length(sys);
  Noptions = length(options);
  %% begin function body
```

```
%% building output depending on options
if Noptions==1
  sys_out(:,1) = reshape(sys,Nsys,1);
else
  for j=1:Noptions
    sys_out(:,j)=copy(reshape(sys,Nsys,1),true);
  end
end
for i_sys=1:Nsys
  for i_options=1:Noptions
    erase_inputs  = find(options(i_options), 'ERASE_INPUTS');
    erase_states  = find(options(i_options), 'ERASE_STATES');
    erase_outputs = find(options(i_options), 'ERASE_OUTPUTS');
    for i=1:length(erase_inputs)
      [pos, sum, value] = ssm.cellstrfind(sys_out(i_sys, i_options).inputnames, erase_inputs{i},'all');
      if sum==0
        error(['input to delete named ',erase_inputs{i}, ' was not found in system ', sys_out(i_sys, i_options).name])
      elseif sum>1
        error(['input to delete named ',erase_inputs{i}, ' was found more than once in system ', sys_out(i_sys, i_options).name])
      else
        Ninputs = sys_out(i_sys, i_options).Ninputs;
        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) ] ;
        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) ] ;
        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) ] ;
        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) ] ;
        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)   ];
        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)];
        sys_out(i_sys, i_options).inputsizes    = [];
      end
    end
    for i=1:length(erase_outputs)
      [pos, sum, value] = ssm.cellstrfind(sys_out(i_sys, i_options).outputnames, erase_outputs{i},'all');
      if sum==0
        error(['input to delete named ',erase_outputs{i}, ' was not found in system ', sys_out(i_sys, i_options).name])
      elseif sum>1
        error(['input to delete named ',erase_outputs{i}, ' was found more than once in system ', sys_out(i_sys, i_options).name])
      else
        Noutputs = sys_out(i_sys, i_options).Noutputs;
        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, :)   ];
        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, :)   ];
        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, :)   ];
        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, :)   ];
        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)   ];
        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)];
        sys_out(i_sys, i_options).outputsizes    = [];
      end
    end
    for i=1:length(erase_states)
      [pos, sum, value] = ssm.cellstrfind(sys_out(i_sys, i_options).ssnames, erase_states{i},'all');
      if sum==0
        error(['input to delete named ',erase_states{i}, ' was not found in system ', sys_out(i_sys, i_options).name])
      elseif sum>1
        error(['input to delete named ',erase_states{i}, ' was found more than once in system ', sys_out(i_sys, i_options).name])
      else
        Nss = sys_out(i_sys, i_options).Nss;
        sys_out(i_sys, i_options).amats = [...
```

```
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
%%%
```

SUBFUNCTIONS

- function ii = getInfo(varargin)
- function plo = getDefaultPlist(set)

SOURCE CODE ^

```
% REDUCE_MODEL enables to do model simplification
%%%
```

```
  end

  % Decide on a deep copy or a modify, depending on the output
  sys = copy(sys, nargout);

  %% begin function body

  % Loop over input systems
  for i_sys=1:Nsys
    for i_options=1:Noptions
      options(i_options) = combine(options(i_options),getDefaultPlist('Default'));
      inputs  = find(options(i_options), 'inputs');
      outputs = find(options(i_options), 'outputs');
      states  = find(options(i_options), 'states');

      % Support single string inputs and single number inputs
      if ischar(inputs)  || isnumeric(inputs),  inputs  = {inputs};  end
      if ischar(outputs) || isnumeric(outputs), outputs = {outputs}; end
      if ischar(states)  || isnumeric(states),  states  = {states};  end

      %% converting the lists of the i/o wanted - ss is not modified
      %                                                    cell_mat   heigths
widths           lines wanted    cols wanted     line names          col names
      [As]                                = ssm.cell_select(sys.amats, sys.sssizes,
sys.sssizes,     states,         states,         sys.ssvarnames,     sys.ssvarnames);
      [Ms]                                = ssm.cell_select(sys.mmats, sys.sssizes,
sys.sssizes,     states,         states,         sys.ssvarnames,     sys.ssvarnames);
      [Bs, sssizes]                       = ssm.cell_select(sys.bmats, sys.sssizes,
sys.inputsizes,  states,         inputs,         sys.ssvarnames,     sys.inputvarnames);
      [Cs]                                = ssm.cell_select(sys.cmats, sys.outputsizes,
sys.sssizes,     outputs,        states,         sys.outputvarnames, sys.ssvarnames);
      [Ds, outputsizes, inputsizes]  = ssm.cell_select(sys.dmats, sys.outputsizes,
sys.inputsizes,  outputs,        inputs,         sys.outputvarnames, sys.inputvarnames);

      [inputvarnames2]   =    ssm.cell_select(sys.inputvarnames,  1,
sys.inputsizes,  {true},         inputs,         {' '},              sys.inputvarnames);
      [ssvarnames2]      =    ssm.cell_select(sys.ssvarnames,     1,
sys.sssizes,     {true},         states,         {' '},              sys.ssvarnames);
      [outputvarnames2]  =    ssm.cell_select(sys.outputvarnames, 1,
sys.outputsizes, {true},         outputs,        {' '},              sys.outputvarnames);

      sys(i_sys, i_options).amats = As;
      sys(i_sys, i_options).mmats = Ms;
      sys(i_sys, i_options).bmats = Bs;
      sys(i_sys, i_options).cmats = Cs;
      sys(i_sys, i_options).dmats = Ds;
      sys(i_sys, i_options).inputvarnames  = inputvarnames2;
      sys(i_sys, i_options).ssvarnames     = ssvarnames2;
      sys(i_sys, i_options).outputvarnames = outputvarnames2;
      sys(i_sys, i_options).inputsizes  = inputsizes;
      sys(i_sys, i_options).sssizes     = sssizes;
      sys(i_sys, i_options).outputsizes = outputsizes;

      sys(i_sys, i_options).name  = [sys(i_sys, i_options).name, ' reduced'];
      sys(i_sys, i_options).description = [sys(i_sys, i_options).description, '
reduced'];
      sys(i_sys, i_options).addHistory(ssm.getInfo(mfilename), options(i_options) ,
{''}, sys(i_sys, i_options).hist );
    end
  end
  sys.validate;
  if nargout > 0
    varargout{1} = sys;
  end
end

%-------------------------------------------------------------------------
% Get Info Object
%-------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v
1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
```

```
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = plist('inputs','ALL','outputs','ALL', 'states', 'ALL');
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
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
%%%
```

```
%%%
```

```
% 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
%%%
```

```
      if sum==1
        aos_pos(positions,:) = [i,j];
        aos_data{positions} = aos_in(positions).data.getY ;
      elseif sum>1
        error(['more than one so variable is named ', value]);
      end
      % getting position
      [positions, sum, value] = ssm.cellstrfind(constants_varnames,sys(i_sys).inputvarnames{i}{j},'all');
      % checking against redundant input data
      if sum==1
        constants_pos(positions,:) = [i,j];
      elseif sum>1
        error(['more than one constant variable is named ', value]);
      end
    end
  end
  %% getting correct number of samples
  Nsamples = find(plist_inputs, 'Nsamples');
  f0 = 1/timestep;
  for i=1:Naos_in
    Nsamples = min(Nsamples,length(aos_in(i).data.y));
    if ~(f0==aos_in(i).data.fs)
      str = ['WARNING : ssm frequency is ',num2str(f0),...
        ' but sampling frequency of ao named ',...
        aos_in(i).name, ' is ', num2str(aos_in(i).data.fs) ];
      utils.helper.msg(utils.const.msg.MNAME, str);
    end
    % maybe tdata should be retrieved and verified to be equal, rather than this.
  end
  if Nsamples == inf % case there is no input!
    Nsamples = 0;
  end

  %% generating random data
  [V,E]=eig(noise_in);
  noise_mat = (V*(E'.^0.5)*V');
  noise_array = noise_mat*randn(Nnoise, Nsamples);
  % How to generate initial position
  % Do one step with firts noise vector
  % muliply inputs by (I - Minv*A)^-1
  % add in the fashion
  %
  %% evaluating data to return
  return_states = find(plist_inputs, 'return states');
  returnStates = false(1,Nss);
  for i=1:Nss
    returnStates(i) = ~isempty(find( ismember(return_states, i)));
  end
  return_outputs = find(plist_inputs, 'return outputs');
  returnOutputs = false(1, Noutputs);
  for i=1:Noutputs
    returnOutputs(i) = ~isempty(find( ismember(return_outputs, i)));
  end
  %% simulation first step
  x = cell(Nss, 1);
  lastX = ssini;
  for i=1:Nss
    if returnStates(i)
      x{i} = zeros(sssizes(i), Nsamples);
    end
  end
  y = cell(Noutputs, 1);
  for i=1:Noutputs
    if returnOutputs(i)
      y{i} = zeros(outputsizes(i), Nsamples);
    end
  end
  IncrementX = cell(Nss,1);
  IncrementY = cell(Noutputs,1);
  %% simulation loop
  for k = 1:Nsamples
    if rem(k,500)==0
       utils.helper.msg(utils.const.msg.MNAME, ['simulation time : ',num2str(k*timestep)]);
    end
    %% resetting each field
    for j = 1:Nss
      IncrementX{j} = zeros(sssizes(j),1);
    end
    for j = 1:Noutputs
      IncrementY{j} = zeros(outputsizes(j),1);
    end
    %% evaluating differences
    for i=1:Nnoise %Noise  MUST BE MODIFIED IF IT IS REALLY SPECIFIED FOR TIME CONTINUOUS SYSTEM!!!!!!
```

```
      for s=1:Nss
        if ~ isempty(sys(i_sys).bmats{s,noise_pos(i,1)})
          IncrementX{s} = IncrementX{s} + sys(i_sys).bmats{s,noise_pos(i,1)}(:,noise_pos(i,2))*noise_array(i,k);
        end
      end
      for j = 1:Noutputs
        if ~ isempty(sys(i_sys).dmats{j,noise_pos(i,1)})
          IncrementY{j} = IncrementY{j} + sys(i_sys).dmats{j,noise_pos(i,1)}(:,noise_pos(i,2))*noise_array(i,k);
        end
      end
    end

    for i=1:Naos_in
      for s=1:Nss
        if ~ isempty(sys(i_sys).bmats{s,aos_pos(i,1)})
          IncrementX{s} = IncrementX{s} + sys(i_sys).bmats{s,aos_pos(i,1)}(:,aos_pos(i,2))*aos_data{i}(k);
        end
      end
      for j = 1:Noutputs
        if ~ isempty(sys(i_sys).dmats{j,aos_pos(i,1)})
          IncrementY{j} = IncrementY{j} + sys(i_sys).dmats{j,aos_pos(i,1)}(:,aos_pos(i,2))*aos_data{i}(k);
        end
      end
    end

    for i=1:Nconstants
      for s=1:Nss
        if ~ isempty(sys(i_sys).bmats{s,constants_pos(i,1)})
          IncrementX{s} = IncrementX{s} + sys(i_sys).bmats{s,constants_pos(i,1)}(:,constants_pos(i,2))*constants_in(i);
        end
      end
      for j = 1:Noutputs
        if ~ isempty(sys(i_sys).dmats{j,constants_pos(i,1)})
          IncrementY{j} = IncrementY{j} + sys(i_sys).dmats{j,constants_pos(i,1)}(:,constants_pos(i,2))*constants_in(i);
        end
      end
    end
    %% computing and storing outputs
    %differential equation
    lastX = ssm.cell_mult(Minv, ssm.cell_add(IncrementX, ssm.cell_mult(sys(i_sys).amats, lastX)));
    %output equations
    lastY = ssm.cell_add(ssm.cell_mult(sys(i_sys).cmats, lastX) ,IncrementY);
    for s=1:Nss
      if returnStates(s)
        x{s}(:,k) = lastX{s};
      end
    end
    for o=1:Noutputs
      if returnOutputs(o)
        y{o}(:,k) = lastY{o};
      end
    end
  end
  %% saving in aos
  Naos = 0;
  for i=1:Nss
    if returnStates(i)
      Naos = Naos +sssizes(i);
    end
  end
  for i=1:Noutputs
    if returnOutputs(i)
      Naos = Naos +outputsizes(i);
    end
  end
  if Naos>0
    ao_out(Naos) = ao;
    if ~exist('t','var')
      t = (1:Nsamples)*timestep;
    end
    count = 1;
    for i=1:Nss
      if returnStates(i)
        for j=1:sssizes(i)
          ao_out(count) = ao(tsdata(t,x{i}(j,:)));
          ao_out(count).setName(['sim_X_',num2str(i),'_',num2str(j)]);
          ao_out(count).setDescription(...
            ['ssm#: ',num2str(i_sys),...
            ', state: ',sys(i_sys).ssnames{i},' ',sys(i_sys).ssvarnames{i}{j}]);
```

```
            ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs, {''}, sys(i_sys).hist );
            count = count+1;
          end
        end
      end
      for i=1:Noutputs
        if returnOutputs(i)
          for j=1:outputsizes(i)
            ao_out(count) = ao(tsdata(t,y{i}(j,:)));
            ao_out(count).setName(['sim_Y_',num2str(i),'_',num2str(j)]);
            ao_out(count).setDescription(...
              ['ssm: ',num2str(i_sys),...
              ', output: ',sys(i_sys).outputnames{i},' variable: ',sys(i_sys).outputvarnames{i}{j}]);
            ao_out(count) = ao_out(count).addHistory(ssm.getInfo(mfilename), plist_inputs, {''}, sys(i_sys).hist );
            count = count+1;
          end
        end
      end
    end
    %% construct output analysis object
    plist_out = plist('lastX', lastX);
    varargout  = {ao_out plist_out};
  end

  function ii = getInfo(varargin)
    if nargin == 1 && strcmpi(varargin{1}, 'None')
      sets = {};
      pls  = [];
    elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
      sets{1} = varargin{1};
      pls = getDefaultPlist(sets{1});
    else
      sets = {'Default'};
      pls = [];
      for kk=1:numel(sets)
        pls = [pls getDefaultPlist(sets{kk})];
      end
    end
    % Build info object
    ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
  end

  function plo = getDefaultPlist(set)
    switch set
      case 'Default'
        plo = plist(...
          'noise variable names',   {}, ...
          'variance',               [], ...
          'aos variable names',     {}, ...
          'aos',                    [], ...
          'constant variable names',{}, ...
          'constant',               [], ...
          'return states',          [], ...
          'return outputs',         1,  ...
          'Nsamples',               1e4, ...
          'ssini',                  {} ...
          );
      otherwise
        error('### Unknown parameter set [%s].', set);
    end
  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
                  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

 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 ^

```
% ssm2iirpz converts a statespace model object to an miir or a pzmodel
%%%
```

```
  end

  % Decide on a deep copy or a modify, depending on the output
  sys = copy(sys, nargout);

  %% begin function body

  %% convert to double
  % Convert to double arrays
  [A,B,C,D,Ts,inputvarnames,ssvarnames,outputvarnames] = double(sys, options);


  Ninputs_out  = numel(inputvarnames);
  Noutputs_out = numel(outputvarnames);

  if Ts > 0
    %% convert to miir
    miir_out(Noutputs_out, Ninputs_out ) = miir();
    for i_inputs=1:Ninputs_out
      for i_outputs=1:Noutputs_out
        sys_loc =ss( A,B(:,i_inputs),C(i_outputs,:),D(i_outputs, i_inputs ),Ts);
        sys_loc = minreal(sys_loc);
        [A_loc, B_loc, C_loc, D_loc]=ssdata(sys_loc);
        [a,b] = ss2tf(A_loc, B_loc, C_loc, D_loc);
        name = [ssm_invars{1},' : ',inputvarnames{i_inputs},' ->
',outputvarnames{i_outputs}];
        if b==0
          m = miir();
        else
          m = miir(real(a), real(b), 1/sys.timestep);
        end
        m.addHistory(ssm.getInfo(mfilename), options , ssm_invars, sys.hist );
        m.name = name;
        miir_out(i_outputs, i_inputs) = m;
      end
    end
    varargout = {miir_out};
  else
    %% convert to pzm
    pzm_out(Noutputs_out, Ninputs_out) = pzmodel();
    for i_inputs=1:Ninputs_out
      for i_outputs=1:Noutputs_out
        [b,a] = ss2tf(A,B(:,i_inputs),C(i_outputs,:),D(i_inputs, i_outputs));
        p = roots(b);
        p = roots2poly(p);
        z = roots(a);
        z = roots2poly(z);
        Gain = a(1)/b(1);
        name = [ssm_invars,'_',num2str(i_inputs),':',num2str(i_outputs)];
        p = pzmodel(Gain,p,z,name);
        p.addHistory(ssm.getInfo(mfilename), options , ssm_invars, sys.hist );
        pzm_out(i_outputs,i_inputs) = p;
      end
    end
    varargout = {pzm_out};
  end

end


function varargout = roots2poly(roots)
  poly = pz();
  i=1;
  while i < length(roots)+1
    if isreal(roots(i))
      poly = [poly pz(roots(i))]; %#ok<AGROW>
      i = i+1;
    else
      f = norm(roots(i))*sign(real(roots(i)));
      Q = norm(roots(i))/abs(2*real(roots(i)));
      poly = [poly pz(f,Q)]; %#ok<AGROW>
      i = i+2; %#ok<FXSET>
    end
  end
  varargout = {poly};
end


%-----------------------------------------------------------------------------
% Get Info Object
%-----------------------------------------------------------------------------
function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
```

```
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', utils.const.categories.statespace, '$Id: resp.m,v 1.17 2008/07/22 10:22:38 ingo Exp $', sets, pls);
end

%--------------------------------------------------------------------------
% Get Default Plist
%--------------------------------------------------------------------------
function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = ssm.getInfo('reduce_model', 'Default').plists;
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
end

```

*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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ssm2ss(varargin)

%% starting initial checks
utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

% Check if this is a call for parameters
if utils.helper.isinfocall(varargin{:})
  varargout{1} = getInfo(varargin{3});
  return
end

% checking number of inputs work data
if ~nargin==2, error('wrong number of inputs!'), end

if ~isequal(class(varargin{1}),'ssm'), error(['argument is not a ssm but a ',
class(varargin{1})]),end
if ~isequal(class(varargin{2}),'plist'), error(['argument is not a plist but a ',
class(varargin{2})]),end

obj = varargin{1};
options = combine(varargin{2}, getDefaultPlist('Default'));

%% begin function body
if ~obj.isnumerical
  error('system should be numeric')
end

[A,B,C,D,Ts,inputvarnames,ssvarnames, outputvarnames] = double(obj, options);
sys = ss(A,B,C,D,Ts);
sys.StateName = ssvarnames;
sys.InputName = inputvarnames;
sys.OutputName = outputvarnames;
sys.Notes = obj.description;
sys.Name = obj.name;
varargout{1} = sys;

end


function ii = getInfo(varargin)
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pls  = [];
  elseif nargin == 1 && ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
```

```
    pls = getDefaultPlist(sets{1});
  else
    sets = {'Default'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'ssm', '', 'STATESPACE', '$Id: ssm2ss.m,v 1.18 2008/08/26 20:55:27 adrien Exp $', sets, pls);
end

function plo = getDefaultPlist(set)
  switch set
    case 'Default'
      plo = ssm.getInfo('reduce_model', 'Default').plists;
    otherwise
      error('### Unknown parameter set [%s].', set);
  end
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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)
       ntaps     - 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', ...
```

```
                              'order', 1,          ...
                              'gain',  2.0,        ...
                              'fs',    10,         ...
                              'fc',    0.2);
              >> f = mfir(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 mfir constructor:
                   >> info = mfir.getInfo
              or   >> info = mfir.getInfo('mfir')

              You can get information about class methods by calling:
                   >> info = mfir.getInfo(method)
              e.g. >> info = mfir.getInfo('eq')

              You can also restrict the sets of parameters contained in
              the minfo object by calling:
                   >> info = mfir.getInfo(method, set)
              e.g. >> info = mfir.getInfo('mfir', 'From Pzmodel')

VERSION:      $Id: mfir.m,v 1.63 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:     miir, ltpda_filter, ltpda_uoh, ltpda_uo, ltpda_obj, plist

%%%
```

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 ^

```
% 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)
%       ntaps     - 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', ...
%                            'order', 1,         ...
```

```
%                             'gain',  2.0,        ...
%                             'fs',    10,         ...
%                             'fc',    0.2);
%               >> f = mfir(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 mfir constructor:
%                    >> info = mfir.getInfo
%               or   >> info = mfir.getInfo('mfir')
%
%               You can get information about class methods by calling:
%                    >> info = mfir.getInfo(method)
%               e.g. >> info = mfir.getInfo('eq')
%
%               You can also restrict the sets of parameters contained in
%               the minfo object by calling:
%                    >> info = mfir.getInfo(method, set)
%               e.g. >> info = mfir.getInfo('mfir', 'From Pzmodel')
%
% VERSION:      $Id: mfir.m,v 1.63 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:     miir, ltpda_filter, ltpda_uoh, ltpda_uo, ltpda_obj, plist
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef mfir < ltpda_filter

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    gd      = [];
    version = '$Id: mfir.m,v 1.63 2008/09/07 10:27:46 hewitson Exp $';
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected, Dependent = true)
    ntaps
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %---------- Abstract Properties ----------
  properties (Abstract = true, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                             Check property setting                        %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods
    function obj = set.gd(obj, val)
      if ~isempty(val)
        if ~isnumeric(val) || ~isreal(val)
          error('### The value for the property ''gd'' must be a real number(s)');
        end
      end
      obj.gd = val;
    end
    function obj = set.ntaps(obj, val)
      error('### Don''t set the property ''ntaps''. It is computed by length(a).');
    end
  end

```

```
%%%
```

```
            a         = find(pl, 'a');
            aoin      = find(pl, 'AO');

            % Selection of construction method
            if ~isempty(filename)
              %--- Construct from file
              utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
              obj = fromFile(obj, pl);
            elseif ~isempty(hostname) || ~isempty(conn)
              utils.helper.msg(msg.OPROC1, 'constructing from repository %s', hostname);
              %--- Construct from repository
              % do hostname constructor
              obj = obj.fromRepository(pl);
            elseif ~isempty(type)
              utils.helper.msg(msg.OPROC1, 'constructing standard %s filter', type);
              %--- Construct from standard type
              obj = fromStandard(obj, pl);
            elseif ~isempty(pzm)
              utils.helper.msg(msg.OPROC1, 'constructing from pzmodel %s', pzm.name);
              %--- Construct from pzmodel
              obj = fromPzmodel(obj, pl);
            elseif ~isempty(a)
              utils.helper.msg(msg.OPROC1, 'constructing from A/B coefficients');
              %--- Construct from a,b coeffs
              obj = fromA(obj, pl);
            elseif ~isempty(aoin)
              utils.helper.msg(msg.OPROC1, 'constructing from AO %s', aoin.name);
              %--- Construct from AO
              obj = fromAO(obj, pl);
            elseif ~isempty(ipl)
              %--- Construct from plist
              % if the plist is empty, we return an empty MFIR
              if nparams(ipl) == 0
              else
                % do plist constructor
                obj = mfir(ipl);
              end
            else
              %--- ERROR
              %%% is the plist is empty then return an empty MFIR
              if nparams(pl) == 0
              else
                error('### Unknown MFIR constructor method.');
              end
            end
          else
            error('### Unknown 1 argument constructor.');
          end
        case 2
          %%%%
```

```
    end % End constructor
  end % End public methods

  %%%%
```

```
0492
0493   methods
0494     varargout = char(varargin);
0495     varargout = copy(varargin)
0496     varargout = display(varargin);
0497     varargout = string(varargin);
0498   end
0499
0500   %%%%
```

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.
%%%
```

```
% 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
%
%%%
```

```
%
%%%
```

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:
```

```
%               >> 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
%
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = copy(old, deepcopy)

  if deepcopy
    t0   = time;
    prov = provenance;
    s    = size(old);
```

```
    obj(s(1),s(2)) = mfir;
    for kk=1:numel(old)
      utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
      obj(kk).gd      = old(kk).gd;
      obj(kk).fs      = old(kk).fs;
      obj(kk).infile  = old(kk).infile;
      obj(kk).a       = old(kk).a;
      obj(kk).histout = old(kk).histout;
      obj(kk).hist    = old(kk).hist;
      obj(kk).name    = old(kk).name;
      % reset on copy
      obj(kk).created = t0;
      obj(kk).creator = prov;
      obj(kk).version = mfir.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

char

PURPOSE ^

CHAR convert a mfir object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a mfir object into a string.
%%%
```

```
%
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for mfir objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for mfir objects.
%%%
```

```
%
%               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
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
function plo = getDefaultPlist()
  plo = plist();
end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

resp

PURPOSE ^

RESP Make a frequency response of the filter.

SYNOPSIS ^

function [varargout] = resp(varargin)

DESCRIPTION ^

```
 RESP Make a frequency response of the filter.
%%%
```

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [varargout] = resp(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% get input filter
  filt = varargin{1};
  if ~isa(filt, 'mfir')
    error('### first input must be an mfir filter object.');
  end

  %%% Collects parameters
  % create empty input if it is not specified
  if nargin == 1
    pl = plist();
  else
    pl = varargin{2};
  end

  % parse input parameter list
  if ~isempty(find(pl, 'f'))
    pl = combine(pl, getDefaultPlist('List'));
  else
    pl = combine(pl, getDefaultPlist('Range'));
  end

  % fill parameter list
  reshape_f = false;
  if isempty(find(pl, 'f'))
    % Compute from frequency range
    f1 = find(pl, 'f1');
    f2 = find(pl, 'f2');
    ndata = find(pl, 'nf');
    scale = find(pl, 'scale');

    % Set frequencies
    if f1 < 0
      f1 = filt.fs/1000;
    end
    if f2 < 0
      f2 = filt.fs/2-1/ndata;
```

```
    end
    switch scale
      case 'lin'
        f   = linspace(f1, f2, ndata);
      case 'log'
        f = logspace(log10(f1), log10(f2), ndata);
    end
  else
    % Compute from frequency list
    f     = find(pl, 'f');
    % Want to deal with rows
    reshape_f = false;
    if size(f,1) > 1
      f = f.';
      reshape_f = true;
    end
    ndata = length(f);
  end

  % create output parameter list, with the frequency array values and the
  % input filter
  plo = plist('f', f, 'filter', filt);

  %%% compute Laplace vector
  s = -1i.*2*pi.*f;

  %%% Compute filter response
  dresp = zeros(1, ndata);

  for n=1:length(filt.a)
    dresp = dresp + filt.a(n).*exp(s.*(n-1)/filt.fs);
  end

  %%% Create an analysis object
  if reshape_f
    f = f.';
    dresp = dresp.';
  end

  % create new output fsdata
  fs = fsdata(f, dresp, filt.fs);

  % make output analysis object
  b = ao(fs);

  % Add history
  b.addHistory(getInfo, plo, [], filt.hist);

  % set name
  b.setName(sprintf('resp(%s)', filt.name), 'internal');

  % Outputs
  if nargout == 0
    iplot(b)
  end

  if nargout == 1
    varargout{1} = b;
  end
  if nargout > 1
    error('incorrect output arguments');
  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 = {};
    pls   = [];
  elseif nargin == 1&& ~isempty(varargin{1}) && ischar(varargin{1})
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
```

```
  else
    sets = {'List', 'Range'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'mfir', '', utils.const.categories.sigproc, '$Id: resp.m,v 1.20 2008/09/04 15:29:30 ingo Exp $', sets, pls);
end

%%%
```

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)
       ntaps     - 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
```

```
                                  designed should be specified as well.
      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 ^

```
% 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)
%       ntaps     - 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
%                                 designed should be specified as well.
%       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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef miir < ltpda_filter

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                            %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    b       = [];
    histin  = [];
    version = '$Id: miir.m,v 1.75 2008/09/07 10:27:46 hewitson Exp $';
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected, Dependent = true)
    ntaps
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %---------- Abstract Properties ----------
  properties (Abstract = true, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                            Check property setting                           %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
  methods
    function obj = set.histin(obj, val)
      if ~isempty(val)
        if ~isnumeric(val) || ~isreal(val)
          error('### The value for the property ''histin'' must be a real number(s)');
        end
      end
      obj.histin = val;
    end
    function obj = set.b(obj, val)
      if ~isempty(val)
        if ~isnumeric(val) || ~isreal(val)
          error('### The value for the property ''b'' must be a real number(s)');
        end
      end
      obj.b = val;
    end
    function obj = set.ntaps(obj, val)
      error('### Don''t set the property ''ntaps''. It is computed by max(length(a), length(b)).');
    end
  end

  %%%%
```

```
        elseif isa(varargin{1}, 'plist')
          %%%%%%%%%%  f = miir(plist-object)   %%%%%%%%%%
          %----------- plist
          pl       = varargin{1};
          filename = find(pl, 'filename');
          hostname = find(pl, 'hostname');
          conn     = find(pl, 'conn');
          type     = find(pl, 'type');
          pzm      = find(pl, 'pzmodel');
          ipl      = find(pl, 'plist');
          a        = find(pl, 'a');

          % Selection of construction method
          if ~isempty(filename)
            utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
            %--- Construct from file
            obj = fromFile(obj, pl);
          elseif ~isempty(hostname) || ~isempty(conn)
            utils.helper.msg(msg.OPROC1, 'constructing from repository %s', hostname);
            %--- Construct from repository
            % do hostname constructor
            obj = obj.fromRepository(pl);
          elseif ~isempty(type)
            utils.helper.msg(msg.OPROC1, 'constructing from standard %s', type);
            %--- Construct from standard type
            obj = fromStandard(obj, pl);
          elseif ~isempty(pzm)
            utils.helper.msg(msg.OPROC1, 'constructing from pzmodel %s', pzm.name);
            %--- Construct from pzmodel
            obj = fromPzmodel(obj, pl);
          elseif ~isempty(a)
            utils.helper.msg(msg.OPROC1, 'constructing from A/B coefficients');
            %--- Construct from a,b coeffs
            obj = fromAB(obj, pl);
          elseif ~isempty(ipl)
            %--- Construct from plist
            % if the plist is empty, we return an empty MIIR
            if nparams(ipl) == 0
            else
              % do plist constructor
              obj = miir(ipl);
            end
          else
            %--- ERROR
            %%% is the plist is empty then return an empty MIIR
            if nparams(pl) == 0
            else
              error('### Unknown MIIR constructor method.');
            end
          end
        elseif isa(varargin{1}, 'pzmodel')
          %%%%%%%%%%  f = miir(pzmodel-object)   %%%%%%%%%%
          %--- Construct from pzmodel
          utils.helper.msg(msg.OPROC1, 'constructing from pzmodel %s', varargin{1}.name);
          obj = fromPzmodel(obj, plist('pzmodel', varargin{1}));

        else
          error('### Unknown single argument constructor.');
        end
      elseif nargin == 2
        %%%%%%%%%%
```

```
      elseif nargin == 3
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   three input   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        utils.helper.msg(msg.OPROC1, 'constructing from A/B coefficients');
        %%%%%%%%%%  f = miir(a, b, fs)   %%%%%%%%%%
        % a,b,fs constructor
        obj = fromAB(obj, plist('a', varargin{1}, 'b', varargin{2}, 'fs', varargin{3}));
      else
          [iirs, invars, rest] = utils.helper.collect_objects(args, 'miir');

          %%% Do we have a list of AOs as input
          if ~isempty(iirs) && isempty(rest)
            obj = miir(iirs);
          else
            error('### Unknown number of constructor arguments.');
          end
      end
    end % End constructor

  end % End public methods

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (static)                                %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Static)

    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: miir.m,v 1.75 2008/09/07 10:27:46 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'miir');
    end

    function out = SETS()
      out = {...
        'Default',            ...
        'From XML File',      ...
        'From MAT File',      ...
        'From LISO File',     ...
        'From Repository',    ...
        'From Standard Type', ...
        'From Pzmodel',       ...
        'From AB',            ...
        'From Plist'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
          %--- From a,b coeffs
        case 'From AB'
          out = plist('a', [], 'b', [], 'fs', []);
          %--- Repository constructor
        case 'From Repository'
          out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
          %--- Create from description
        case 'From Standard Type'
          out = plist('type','lowpass', 'fc', 0.1, ...
            'gain',  1.0, 'fs', 1.0, 'order', 1, 'ripple', 0.5);
          %--- Create from a pzmodel
        case 'From Pzmodel'
          out = plist('pzmodel', pzmodel());
          %--- Read from XML file
        case 'From XML File'
          out = plist('filename', '');
          %--- Read from MAT file
        case 'From MAT File'
          out = plist('filename', '');
          %--- Create from LISO file
        case 'From LISO File'
          out = plist('filename', '');
          %--- Create from a plist
        case 'From Plist'
          out = plist('Plist', []);
        otherwise
          out = plist();
      end
    end % function out = getDefaultPlist(varargin)

```

```
0479
0480    end % End static methods
0481
0482    %%%%
```

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 %
```

```
% 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
%
%%%
```

```
%
% FUNCTION:    getInfo
%
% DESCRIPTION: Get Info Object
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%
```

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
%%%
```

```
    prov = provenance;
    s    = size(old);
    obj(s(1),s(2)) = miir;
    for kk=1:numel(old)
      utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
      obj(kk).fs      = old(kk).fs;
      obj(kk).infile  = old(kk).infile;
      obj(kk).a       = old(kk).a;
      obj(kk).b       = old(kk).b;
      obj(kk).histin  = old(kk).histin;
      obj(kk).histout = old(kk).histout;
      obj(kk).hist    = old(kk).hist;
      obj(kk).name    = old(kk).name;
      obj(kk).created = t0;
      obj(kk).creator = prov;
      obj(kk).version = miir.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setHistin

PURPOSE ^

SETHISTIN Set the property 'histin'

SYNOPSIS ^

function varargout = setHistin(varargin)

DESCRIPTION ^

```
 SETHISTIN Set the property 'histin'
%%%
```

```
%               >> 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
%
%%%
```

char

PURPOSE ^

CHAR convert a miir object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a miir object into a string.
%%%
```

```
%
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for miir objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for miir objects.
%%%
```

```
%
%               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
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
%%%
```

resp

PURPOSE ^

RESP Make a frequency response of the filter.

SYNOPSIS ^

function varargout = resp(varargin)

DESCRIPTION ^

```
 RESP Make a frequency response of the filter.
%%%
```

- function ii = getInfo(varargin)
- function plo = getDefaultPlist(set)

SOURCE CODE ^

```
% RESP Make a frequency response of the filter.
%%%
```

```
  if isempty(find(pl, 'f'))
    % Compute from frequency range
    f1 = find(pl, 'f1');
    f2 = find(pl, 'f2');
    ndata = find(pl, 'nf');
    scale = find(pl, 'scale');

    % Set frequencies
    if f1 < 0
      f1 = filt.fs/1000;
    end
    if f2 < 0
      f2 = filt.fs/2-1/ndata;
    end
    switch scale
      case 'lin'
        f   = linspace(f1, f2, ndata);
      case 'log'
        f = logspace(log10(f1), log10(f2), ndata);
    end
  else
    % Compute from frequency list
    f     = find(pl, 'f');
    % Want to deal with rows
    reshape_f = false;
    if size(f,1) > 1
      f = f.';
      reshape_f = true;
    end
    ndata = length(f);
  end

  % create output parameter list, with the frequency array values and the
  % input filter
  plo = plist('f', f, 'filter', filt);

  %%% compute Laplace vector
  s = -1i.*2*pi.*f;

  %%% Compute filter response
  num = zeros(1, ndata);
  for n=1:length(filt.a)
    num = num + filt.a(n).*exp(s.*(n-1)/filt.fs);
  end
  denom = zeros(1, ndata);
  for n=1:length(filt.b)
    denom = denom + filt.b(n).*exp(s.*(n-1)/filt.fs);
  end
  dresp = num ./ denom;

  % mag   = 20*log10(abs(dresp));
  % phase = angle(dresp)*180/pi;

  %%% Create an analysis object
  if reshape_f
    f = f.';
    dresp = dresp.';
  end

  % create new output fsdata
  fs = fsdata(f, dresp, filt.fs);

  % make output analysis object
  b = ao(fs);

  % Add history
  b.addHistory(getInfo, plo, [], filt.hist);

  % set name
  b.setName(sprintf('resp(%s)', filt.name), 'internal');

  % Outputs
  if nargout == 0
    iplot(b)
  end

  if nargout == 1
    varargout{1} = b;
  end
  if nargout > 1
    error('incorrect output arguments');
  end
end

%%%
```

```
%%%
```

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.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = string(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  objs = utils.helper.collect_objects(varargin(:), 'pzmodel');

  cmd = '';
  for kk = 1:numel(objs)

    if ~isempty(objs(kk).hist)
      cmd = [cmd, sprintf('pzmodel(%s), ', string(objs(kk).hist.plistUsed))];
    else
      pl = plist();
      fields = fieldnames(objs(kk));
      for ii = 1:length(fields)
        field = fields{ii};
        pl = append(pl, field, objs(kk).(field));
      end

      cmd = [cmd 'pzmodel(' string(pl) '), '];
    end

  end

  cmd = cmd(1:end-2);

  if numel(objs) > 1
    cmd = ['[' cmd ']'];
  end

  varargout{1} = cmd;

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, 'pzmodel', '', utils.const.categories.helper, '$Id: string.m,v 1.10 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function plo = getDefaultPlist()
  plo = plist();
end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 of copy

```
function varargout = copy(old, deepcopy)

  if deepcopy
    t0   = time;
    prov = provenance;
    s = size(old);
    obj(s(1),s(2)) = pzmodel;
    for kk=1:numel(old)
      utils.helper.msg(utils.const.msg.OPROC2, 'copying %s', old(kk).name);
      obj(kk).gain  = old(kk).gain;
      if ~isempty(old(kk).poles)
        obj(kk).poles = copy(old(kk).poles,1);
      else
        obj(kk).poles = [];
      end
      if ~isempty(old(kk).zeros)
        obj(kk).zeros = copy(old(kk).zeros,1);
      else
        obj(kk).zeros = [];
      end
      obj(kk).hist  = old(kk).hist;
      obj(kk).name  = old(kk).name;
      obj(kk).created  = t0;
      obj(kk).creator  = prov;
      obj(kk).version  = pzmodel.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %
```

```
% VERSION:     $Id: getlowerFreq.m,v 1.7 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     04-04-2007 M Hewitson
%                 Creation
%
%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
%               >> 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
%
%%%
```

*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 %
```

```
%               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
%
%%%
```

```
      utils.helper.msg(msg.OPROC2, 'computing complex pole');
      [ai,bi] = cp2iir(cpoles(j), fs);
      if ~isempty(ao)>0
        [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
      else
        ao = ai;
        bo = bi;
      end
    end
  else
    % do remaining czeros
    for j=length(cpoles)+1:length(czeros)
      utils.helper.msg(msg.OPROC2, 'computing complex zero');
      [ai,bi] = cz2iir(czeros(j), fs);
      if ~isempty(ao)>0
        [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
      else
        ao = ai;
        bo = bi;
      end
    end
  end

  % Now do the real poles and zeros
  for j=1:np
    pole = poles(j);
    if isnan(pole.q) || pole.q < 0.5
      utils.helper.msg(msg.OPROC2, 'computing real pole');
      [ai,bi] = rp2iir(pole, fs);
      if ~isempty(ao)>0
        [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
      else
        ao = ai;
        bo = bi;
      end
    end
  end

  for j=1:nz
    zero = zeros(j);
    if isnan(zero.q) || zero.q < 0.5
      utils.helper.msg(msg.OPROC2, 'computing real zero');
      [ai,bi] = rz2iir(zero, fs);
      if ~isempty(ao)>0
        [ao,bo] = pzmodel.abcascade(ao,bo,ai,bi);
      else
        ao = ai;
        bo = bi;
      end
    end
  end

  ao = ao.*gain;

  varargout{1} = ao;
  varargout{2} = bo;
end

%%%
```

```
  a(1) = (4*fs*fs + 2*wz*fs/zq + wz2)/k;
  a(2) = (2*wz2 - 8*fs*fs)/k;
  a(3) = (4*fs*fs - 2*wz*fs/zq + wz2)/k;
  b(1) = 1;
  b(2) = (2*wp2 - 8*fs*fs)/k;
  b(3) = (wp2 + 4*fs*fs - 2*wp*fs/pq)/k;

  % normalise dc gain to 1
  g = iirdcgain(a,b);
  a = a / g;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    cpolezero
%
% DESCRIPTION: Work out the DC gain of an IIR filter given the coefficients.
%
% CALL:        g = iirdcgain(a,b)
%
% INPUTS:      a - numerator coefficients
%              b - denominator coefficients
%
% OUTPUTS      g - gain
%
% HISTORY:     03-07-2002 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function g = iirdcgain(a,b)
  suma = sum(a);
  if(length(b)>1)
    sumb = sum(b);
    g = suma / sumb;
  else
    g = suma;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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, 'pzmodel', '', utils.const.categories.internal, '$Id: pzm2ab.m,v 1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

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 AO is set accordingly.
              The function then calls mfir() with the new fsdata AO 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 ^

```
% 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 AO is set accordingly.
%              The function then calls mfir() with the new fsdata AO 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = tomfir(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  pl = plist;

  if nargin == 1
    pzm = varargin{1};
  elseif nargin == 2
    pzm = varargin{1};
    pl  = varargin{2};
  else
    error('### Incorrect inputs.');
  end

  % check design parameters
  fs = find(pl, 'fs');

  r = resp(pzm, plist('f1', 0, 'f2', fs/2, 'nf', 1000));

  % Set fs
  r.setFs(fs);

  % compute filter
  f = mfir(r);

  if nargout == 1
    varargout{1} = f;
  else
    error('### Incorrect outputs.');
  end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getInfo
%
% DESCRIPTION: Get Info Object
%
```

```
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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.
%%%
```

```
%
% 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
%
%%%
```

```
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

char

PURPOSE ^

CHAR convert a pzmodel object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a pzmodel object into a string.
%%%
```

```
% HISTORY:      02-04-2007 M Hewitson
%                  Creation
%
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for pzmodel objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for pzmodel objects.
%%%
```

```
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 %%%%
```

```
  ii = minfo(mfilename, 'pzmodel', '', utils.const.categories.output, '$Id: display.m,v 1.9 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%
```

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 ^

```
% FNGEN creates an arbitrarily long time-series based on the input pzmodel.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
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 %%%%
```

```
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = resp(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Input objects checks
  if nargin < 1
    error('### incorrect number of inputs.')
  end

  % Collect input pzmodels, plists and input variable names
  in_names = cell(size(varargin));
  for ii = 1:nargin
    in_names{end+1} = inputname(ii);
  end
  [pzms, invars] = utils.helper.collect_objects(varargin(:), 'pzmodel', in_names);

  % Loop over pzmodels
  bs = [];
  for pp=1:numel(pzms)

    % process this pzmodel
    pzm = pzms(pp);
    % Now look at the model
    name  = pzm.name;
    gain  = pzm.gain;
    poles = pzm.poles;
    zeros = pzm.zeros;
    np    = length(poles);
    nz    = length(zeros);

    %%% check inputs

    if nargin == 1
      %%% a = resp(pzm); %%
      % get frequencies for response
      minf = getlowerFreq(pzm)/10;
      maxf = getupperFreq(pzm)*10;
      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
0129      end
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');
0163
```

```
    % Add to outputs
    bs = [bs b];

  end

  % Outputs
  if nargout == 0
    iplot(bs)
  elseif nargout == 1
    varargout{1} = bs;
  else
    error('### incorrect output arguments');
  end

end


%%%
```

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 %%%%%%%%%%%%%%%%%%%%

- function out = VEROUT()
- function ii = getInfo(varargin)
- function out = SETS()
- function out = getDefaultPlist(set)

SOURCE CODE ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef timespan < ltpda_uoh

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                            %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    startT     = time(0);
    endT       = time(0);
    timeformat = time(0).timeformat;
    timezone   = time(0).timezone;
    interval   = '';
    version    = '$Id: timespan.m,v 1.42 2008/09/07 10:48:46 hewitson Exp $';
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %---------- Abstract Properties ----------
  properties (Abstract = true, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                             Check property setting                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = set.startT(obj, val)
      if ~isa(val, 'time')
        error('### The value for the property ''startT'' must be a time-object.\nBut it is from the class [%s]',class(val));
      end
      obj.startT = val;
    end
    function obj = set.endT(obj, val)
      if ~isa(val, 'time')
        error('### The value for the property ''endT'' must be a time-object.\nBut it is from the class [%s]',class(val));
      end
      obj.endT = val;
    end
    function obj = set.timeformat(obj, val)
      if ~ischar(val)
        error('### The value for the property ''timeformat'' must be a character string.');
      end
      obj.timeformat = val;
    end
    function obj = set.timezone(obj, val)
      if ~strcmp(class(val), 'sun.util.calendar.ZoneInfo')
        error('### The value for the property ''timezone'' must be a JAVA sun.util.calendar.ZoneInfo-object.');
      end
      obj.timezone = val;
    end
    function obj = set.interval(obj, val)
```

```
      if ~ischar(val)
        error('### The value for the property ''interval'' must be a character string.');
      end
      obj.interval = val;
    end
  end

  %%%%
```

```
            if isempty(endT)
              endT = find(pl, 'end');
            end

            if ~isempty(filename)
              %-------------------------------------------------------
              %--- Construct from file
              %-------------------------------------------------------

              utils.helper.msg(msg.OPROC1, 'constructing from file %s', filename);
              obj = fromFile(obj, varargin{1});

            elseif ~isempty(hostname) || ~isempty(conn)
              %-------------------------------------------------------
              %--- Construct from repository
              %-------------------------------------------------------
              utils.helper.msg(msg.OPROC1, 'constructing from repository %s', hostname);
              obj = obj.fromRepository(pl);

            elseif ~isempty(startT) && ~isempty(endT)
              %-------------------------------------------------------
              %--- Construct from start and end times
              %-------------------------------------------------------
              utils.helper.msg(msg.OPROC1, 'constructing from start/end times');
              obj = obj.fromTimespanDef(pl);

            elseif ~isempty(ipl)
              %-------------------------------------------------------
              %--- Construct from plist
              %-------------------------------------------------------

              if nparams(ipl) == 0
                %%% return empty object
              else
                obj = timespan(ipl);
              end

            else

              if nparams(pl) == 0
                %%% if the plist is empty then return the default time object
              else
                error('### Unknown TIMESPAN constructor method.');
              end
            end

          elseif isstruct(varargin{1})
            %%%%%%%%%%   ts = time(struct)   %%%%%%%%%%

            utils.helper.msg(msg.OPROC1, 'constructing from struct');
            %%% Set properties which are declared in this class
            ts_struct = varargin{1};

            obj.startT     = utils.helper.struct2obj(ts_struct.startT, 'time');
            obj.endT       = utils.helper.struct2obj(ts_struct.endT, 'time');
            obj.timeformat = ts_struct.timeformat;
            obj.timezone   = ts_struct.timezone;
            obj.interval   = ts_struct.interval;
            obj.version    = ts_struct.version;

          else
            error('### Unknown single argument constructor.');
            end

        case 2
          %%%%%%%%%%
```

```
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            %%%%%%%%%%
```

```
    end % End constructor

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                Methods (public)                            %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods
    varargout = copy(varargin)
    varargout = setStartT(varargin)
    varargout = setEndT(varargin)
    varargout = setTimeformat(varargin)
    varargout = setTimezone(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (protected)                           %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Access = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                             Methods (private)                              %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Access = private)
    obj = fromTimespanDef(obj, pli)
    obj = computeInterval(obj)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (static)                              %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  methods (Static)

    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: timespan.m,v 1.42 2008/09/07 10:48:46 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'timespan');
    end

    function out = SETS()
      out = {...
        'Default',          ...
        'From XML File',    ...
        'From MAT File',    ...
        'From Repository', ...
        'From Timespan Definition', ...
        'From Plist'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        case 'From Repository'
          %--- Repository constructor
          out = plist('hostname', 'localhost', 'database', 'ltpda', 'ID', []);
        case 'From Timespan Definition'
          %--- From definition constructor
          out = plist(...
            'startT', '1970-01-01 00:30:00.000', ...
            'endT',   '1980-01-01 12:00:00.010', ....
            'timezone', 'UTC',                   ...
            'timeformat', 'yyyy-mm-dd HH:MM:SS.FFF');
        case 'From XML File'
```

```
          %--- Read from XML file
          out = plist('filename', '');
        case 'From MAT File'
          %--- Read from MAT file
          out = plist('filename', '');
        case 'From Plist'
          %--- Create from a plist
          out = plist('Plist', []);
        otherwise
          error('### Unknown set-name [%s] for a default parameter list.', set);
      end
    end % function out = getDefaultPlist(varargin)

  end % End static methods

  %%%%
```

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 ^

```
% 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
%
%%%
```

```
  end

  %%% Prepare output
  varargout{1} = objs;

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, 'timespan', '', utils.const.categories.helper, '$Id: setEndT.m,v 1.2 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist('endT', '');
end

```

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 ^

```
% 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
%
%%%
```

```
0095
0096   %%% Prepare output
0097   varargout{1} = objs;
0098
0099 end
0100
0101 %%%%
```

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 ^

```
% 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
%
%%%
```

```
%%%
```

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 ^

```
% 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
%
%%%
```

```
%%%
```

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
%%%
```

```
%%%
```

char

PURPOSE ^

CHAR convert a timespan object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a timespan object into a string.
%%%
```

```
%
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = char(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % Collect all time-objects
  objs = utils.helper.collect_objects(varargin(:), 'timespan');

  % go through time-object
  pstr = '';
  for jj=1:numel(objs)
    pstr = [pstr '(start: ' char(objs(jj).startT) ' end: ' char(objs(jj).endT) '), '];
  end

  varargout{1} = pstr(1:end-2);
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, 'timespan', '', utils.const.categories.output, '$Id: char.m,v 1.5 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:     getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

display

PURPOSE ^

DISPLAY overloads display functionality for timespan objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for timespan objects.
%%%
```

Description of 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
%
%%%
```

```
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 nargout == 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  %%%%
```

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
object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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
```

```
%
% 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
%
%%%
```

```
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

setName

PURPOSE ^

SETNAME Set the property 'name'.

SYNOPSIS ^

function varargout = setName(varargin)

DESCRIPTION ^

```
 SETNAME Set the property 'name'.
%%%
```

```
%                     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
%
%%%
```

```
    %%% set the value
    objs(ii).name = vals;
  end

  %%% Prepare output
  varargout{1} = objs;
end


%%%
```

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.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = append(varargin)

  import utils.const.*

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  utils.helper.msg(msg.OMNAME, 'running %s/%s', mfilename('class'), mfilename);

  [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');
  [pps,  invars, rest] = utils.helper.collect_objects(rest(:), 'param');

  %%% Decide on a deep copy or a modify
  pls = copy(objs, nargout);

  %%% REMARK: If the rest is an single string and the number of plist is two
  %%%         then we assume that the rest and the second plist are a key/value
  %%%         pair.
  if numel(rest) == 1 && ischar(rest{1}) && numel(objs) == 2
    rest{2} = objs(2);
    pls(2) = [];
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%                First Case: Append plist-objects                %%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  pl = pls(1);

  %%% If we found more than one plist then append the parameters
  %%% of the second, third, ... plist to the first plist
  if numel (pls) > 1
    for kk = 2:numel(pls)
      for jj = 1:length(pls(kk).params)
        add_param(pl, pls(kk).params(jj));
      end
    end
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%                Second Case: Append param-objects               %%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  if ~isempty(pps)
    for kk = 1:numel(pps)
      add_param(pl, pps(kk));
    end
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%                Third Case: Append key/value pairs              %%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  if ~isempty(rest)
    if mod(numel(rest),2) ~= 0
      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(1) = [];
0096
0097        add_param(pl, param(key,val));
0098      end
0099    end
0100
0101    varargout{1} = pl;
0102  end
0103
0104
0105  %%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %%%%
```

```
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

find

PURPOSE ^

FIND overloads find routine for a parameter list.

SYNOPSIS ^

function varargout = find(varargin)

DESCRIPTION ^

```
 FIND overloads find routine for a parameter list.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = find(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');

  %%%%%%%%%%   Some plausibility checks   %%%%%%%%%%
  if numel(objs) ~= 1
    error('### This function could only work with one plist-object');
  end

  if numel(rest) ~= 1
    error('### Please specify only one ''key''.');
  end

  if ~ischar(rest{1})
    error('### The ''key'' must be a string but it is from the class %s.',
class(rest{1}));
  end

  pl  = objs(1);
  key = rest{1};
  varargout{1} = [];

  for ii=1:length(pl.params)
    pp = pl.params(ii);
    if strcmpi(pp.key, key)
      if isa(pp.val, 'ltpda_obj')
        varargout{1} = copy(pp.val, 1);
      else
        varargout{1} = pp.val;
      end
      return
    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, 'plist', '', utils.const.categories.helper, '$Id: find.m,v 1.13
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

```

```
%%%
```

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.
%%%
```

```
%
% 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
%
%%%
```

```
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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.
%%%
```

```
%
% HISTORY:      30-01-2007 M Hewitson
%                  Creation
%
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = pset(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');
  [pps,  invars, rest] = utils.helper.collect_objects(rest(:), 'param');

  %%% Decide on a deep copy or a modify
  pls = copy(objs, nargout);

  %%%%%%%%%%   Some plausibility checks   %%%%%%%%%%
  if (isempty(pps) && isempty(rest)) || mod(numel(rest),2)
    error('### Please define a ''key'' AND a ''value''%s### to set this pair.',
char(10));
  end

  for ii = 1:numel(pls)

    %%%%%%%%%%   First case: Set param-objects   %%%%%%%%%%
    if ~isempty(pps)
      for jj = 1:numel(pps)
        add_param(pls(ii), pps(jj));
      end
    end

    %%%%%%%%%%   Second case: Set key/value pair   %%%%%%%%%%
    rest_help = rest;
    while ~isempty(rest_help)
      key = rest_help{1};
      val = rest_help{2};

      %%% Remove the first two objects from the 'rest_help' variable
      rest_help(1) = [];
      rest_help(1) = [];

      add_param(pls(ii), param(key,val));
    end
  end

  % Set output
  if nargout > 0
    varargout{1} = pls;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DESCRIPTION: check to see if this is one of the parameters we can set,
%              otherwise add it.
function add_param(pl, pp)
  found = 0;
  for j=1:length(pl.params)
    if strcmpi(pl.params(j).key, pp.key)
      pl.params(j).setVal(pp.val);
      found = 1;
      break
    end
  end
  % add this parameter if necessary
  if ~found
```

```
    pl.params = [pl.params pp];
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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, 'plist', '', utils.const.categories.helper, '$Id: pset.m,v 1.11
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

remove

PURPOSE ^

REMOVE remove a parameter from the parameter list.

SYNOPSIS ^

function varargout = remove(varargin)

DESCRIPTION ^

```
 REMOVE remove a parameter from the parameter list.
%%%
```

```
%
% VERSION:     $Id: remove.m,v 1.13 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     30-01-2007 M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = remove(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  [objs, invars, rest] = utils.helper.collect_objects(varargin(:), 'plist');

  %%%%%%%%%%   Some plausibility checks   %%%%%%%%%%
  if numel(rest) ~= 1
    error('### Specify only an index OR a key-name to remove the corresponding parameter.');
  end

  val = rest{1};

  for ii = 1:numel(objs)
    pl = objs(ii);
    np = length(pl.params);

    %%%%%   Remove specified key   %%%%%
    if ischar(val)
      idx = [];
      for i=1:np
        key = pl.params(i).key;
        if ~strcmpi(key, val)
          idx = [idx i];
        end
      end
      pl.params = pl.params(idx);
      if length(idx) == np
        warning('!!! No parameter found matching key name in %d. plist', ii);
      end

      %%%%%   Remove specified position   %%%%%
    elseif isnumeric(val)

      idx = [1:val-1 val+1:np];
      if max(idx) > np
        error('### Index exceeds number of parameters in %d. plist', ii);
      end
      pl.params = pl.params(idx);

      %%%%%   Unknown method   %%%%%
    else
      error('### unknown indexing method')
    end

    objs(ii) = pl;
  end

  varargout{1} = objs;
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, 'plist', '', utils.const.categories.helper, '$Id: remove.m,v 1.13 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%
```

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
object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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)
```

```
%
% 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
%
%%%
```

```
      end

      % close bracket
      if ~isempty(pstr)
        pstr = pstr(1:end-2);
      end
      pstr = [pstr '), '];
    else
      pstr = [pstr 'plist(), '];
    end
  end

  if numel(objs) > 1
    pstr = ['[ ' pstr(1:end-2) ' ]'];
  else
    pstr = pstr(1:end-2);
  end

  varargout{1} = pstr;
end

%%%
```

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 of copy

```
%%%
```

char

PURPOSE ^

CHAR convert a parameter list into a string.

SYNOPSIS ^

function varargout = char(varargin)

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
%
%%%
```

```
end

```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

display

PURPOSE ^

DISPLAY display plist object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY display plist object.
%%%
```

```
% VERSION:     $Id: display.m,v 1.11 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     30-01-2007 M Hewitson
%                 Creation
%
%%%
```

```
1.11 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%
```

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 %
```

```
% 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
%
%%%
```

```
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 %%%%
```

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
%%%
```

```
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = specwin.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end

```

*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.
%%%
```

```
function varargout = char(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% Get input
  objs = utils.helper.collect_objects(varargin(:), 'specwin');

  pstr = [];
  for kk = 1:numel(objs)
    w = objs(kk);

    pstr = ['(' pstr w.type];
    pstr = [pstr ', alpha=' num2str(w.alpha)];
    pstr = [pstr ', psll=' num2str(w.psll)];
    pstr = [pstr ', rov=' num2str(w.rov)];
    pstr = [pstr ', nenbw=' num2str(w.nenbw)];
    pstr = [pstr ', w3db=' num2str(w.w3db)];
    pstr = [pstr ', flatness=' num2str(w.flatness) '), '];

    pstr = strrep(pstr, '_', '\_');
  end

  varargout{1} = pstr(1:end-2);

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, 'specwin', '', utils.const.categories.output, '$Id: char.m,v 1.7
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

display

PURPOSE ^

DISPLAY overloads display functionality for specwin objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for specwin objects.
%%%
```

```
%
% VERSION:     $Id: display.m,v 1.10 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     30-01-2007 M Hewitson
%                 Creation
%
%%%
```

plot

PURPOSE ^

PLOT plots a specwin object.

SYNOPSIS ^

function varargout = plot(varargin)

DESCRIPTION ^

```
 PLOT plots a specwin object.
%%%
```

```
%
% VERSION:     $Id: plot.m,v 1.9 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     06-02-2007 M Hewitson
%                 Creation
%
%%%
```

```
%
% FUNCTION:     getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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.
%%%
```

```
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 %%%%
```

minus

PURPOSE ^

MINUS overloads - operator for time objects.

SYNOPSIS ^

function varargout = minus(varargin)

DESCRIPTION ^

```
 MINUS overloads - operator for time objects.
%%%
```

```
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 %%%%
```

```
%%%
```

plus

PURPOSE ^

PLUS overloads + operator for time objects.

SYNOPSIS ^

function varargout = plus(varargin)

DESCRIPTION ^

```
 PLUS overloads + operator for time objects.
%%%
```

```
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 %%%%
```

```
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 %%%%
```

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

%%%
```

- 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 ^

```
% TIME time object class constructor.
%%%
```

```
%
% 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
%
%%%
```

```
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.');
0278                end
0279              end
0280
0281            else
0282              error('### Unknown single argument constructor.');
0283            end
0284
0285          case 2
0286            %%%%%%%%%%
```

```
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    %%%%%%%%%%
```

```
            'timeformat',       'yyyy-mm-dd HH:MM:SS.FFF');
        case 'From Time String'
          %--- String constructor
          out = plist(...
            'time_str', '1970-01-01 00:00:00.000', ...
            'timezone',    'UTC', ...
            'timeformat', 'yyyy-mm-dd HH:MM:SS.FFF');
          %--- Create from a plist
        case 'From Plist'
          out = plist('Plist', []);
        otherwise
          error('### Unknown set [%s] to get the default plist.', set);
      end
    end % End getDefaultPlist

  end % End static methods

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                            Methods (static, private)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static, Access=private)

    varargout = parse_time_string(varargin)
    [time_str, created_str] = epoch2str(tt)

  end % End static, private methods
end % End classdef
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %%%%
```

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
```

```
%               >> 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = string(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % collect the time-objects
  tt = utils.helper.collect_objects(varargin(:), 'time');

  cmd = '';
  for jj=1:length(tt)

    pl = plist();
    pl.append('timeformat',      char(tt(jj).timeformat));
    pl.append('timezone',        char(tt(jj).timezone.getID));
    pl.append('utc_epoch_milli', tt(jj).utc_epoch_milli);

    cmd = [cmd 'time(' string(pl) ') '];

  end

  %%% Wrap the command only with brackets if the there are more than one object
  if numel(tt) > 1
    cmd = ['[' cmd(1:end-1) ']'];
  end

  % prepare output
  varargout{1} = cmd;

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, 'time', '', utils.const.categories.helper, '$Id: string.m,v 1.9
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
```

```
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
% HISTORY:      11-07-2008 Diepholz
%                  Creation
%
%%%
```

setEpochtime

PURPOSE ^

SETEPOCHTIME Set the property 'utc_epoch_milli'.

SYNOPSIS ^

function varargout = setEpochtime(varargin)

DESCRIPTION ^

```
 SETEPOCHTIME Set the property 'utc_epoch_milli'.
%%%
```

```
%
% 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
%
%%%
```

```
%
% DESCRIPTION: Get Info Object
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
% FUNCTION:    compute_epoch_t
%
% DESCRIPTION: Compute the epoch time from a time-string.
%
% HISTORY:     07-07-2008 M Diepholz
%                 Creation.
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
end

%%%
```

```
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
  %%% Prepare output
  varargout{1} = objs;

end

%%%
```

char

PURPOSE ^

CHAR convert a time object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a time object into a string.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = char(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % Collect all time-objects
  objs = utils.helper.collect_objects(varargin(:), 'time');

  % go through time-object
  pstr = '';
  for jj=1:numel(objs)
    pstr = [pstr time.epoch2str(objs(jj)) ', '];
  end

  varargout{1} = pstr(1:end-2);
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, 'time', '', utils.const.categories.output, '$Id: char.m,v 1.12
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:     getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

display

PURPOSE ^

DISPLAY overloads display functionality for time objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for time objects.
%%%
```

```
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 %%%%
```

```
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  %%%%
```

```
function plo = getDefaultPlist()
  plo = plist();
end

```

---

*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.
%%%
```

```
%
% VERSION:     $Id: format.m,v 1.11 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     23-07-2007 Diepholz
%                 Creation
%
%%%
```

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 comlpex 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

%%%
```

```
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 comlpex 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 %%%%
```

```
0097    properties (Abstract = true, SetAccess = protected)
0098    end
0099
0100    %%%%
```

```
            obj.setRI(ri);
          else
            error('### Unknown plist constructor');
          end

        elseif isstruct(varargin{1})
          %%%%%%%%%%   p1 = pz(structure)   %%%%%%%%%%
          %%% Set properties which are declared in this class
          obj.f       = varargin{1}.f;
          obj.q       = varargin{1}.q;
          obj.ri      = varargin{1}.ri;
          obj.version = varargin{1}.version;

        elseif iscell(varargin{1})
          %%%%%%%%%%   p1 = pz({[1 2], 1+2i, 3})   %%%%%%%%%%
          obj(1:numel(varargin{1})) = pz();
          for kk = 1:numel(varargin{1})
            obj(kk) = pz(varargin{1}{kk});
          end

        else
          error('### Unknown call of the pz constructor.');
        end

      elseif nargin == 2
        %%%%%%%%%%
```

```
  methods (Static)

    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: pz.m,v 1.23 2008/09/03 16:36:02 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'pz');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          out = plist();
      end
    end

  end

  %%%%
```

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
%%%
```

```


  if deepcopy
    ps = properties('pz');
    s = size(old);
    obj(s(1),s(2)) = pz;
    for kk=1:numel(old)
      utils.helper.msg(utils.const.msg.OPROC2, 'copying pz');
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = pz.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
bilinear transform.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 %
```

```
% VERSION:     $Id: cp2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     03-04-2007 M Hewitson
%                  Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = cp2iir(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  p  = varargin{1};
  fs = varargin{2};

  f0 = p.f;
  q  = p.q;

  w0  = f0*2*pi;
  w02 = w0^2;

  k    = (q*w02 + 4*q*fs*fs + 2*w0*fs) / (q*w02);
  b(1) =  1;
  b(2) = (2*w02-8*fs*fs) / (k*w02);
  b(3) = (q*w02 + 4*q*fs*fs - 2*w0*fs) / (k*q*w02);

  a(1) =  1/k;
  a(2) = -2/k;
  a(3) = -1/k;
  a    =  a*-2;

  varargout{1} = a;
  varargout{2} = b;
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, 'pz', '', utils.const.categories.internal, '$Id: cp2iir.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

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
bilinear transform.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 %
```

```
% VERSION:     $Id: cz2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     03-04-2007 Hewitson
%                 Creation
%
%%%
```

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 ^

```
% 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
%
%%%
```

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
transform.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 %
```

```
% VERSION:     $Id: rz2iir.m,v 1.4 2008/09/04 15:29:31 ingo Exp $
%
% HISTORY:     18-02-2003 Hewitson
%                 Creation
%
%%%
```

char

PURPOSE ^

CHAR convert a pz object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a pz object into a string.
%%%
```

```
function varargout = char(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  objs = utils.helper.collect_objects(varargin(:), 'pz');

  pstr = '';
  for ii = 1:numel(objs)
    pp   = objs(ii);
    % f and Q
    pstr = [pstr  sprintf('(f=%g Hz,Q=%g), ', pp.f, pp.q)];
  end

  %%% Prepare output
  varargout{1} = pstr(1:end-2);
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, 'pz', '', utils.const.categories.output, '$Id: char.m,v 1.5
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end
```

display

PURPOSE ^

DISPLAY display a pz object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY display a pz object.
%%%
```

```
%                  Creation
%
%%%
```

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 ltpda 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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = string(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  p = utils.helper.collect_objects(varargin(:), 'pz');

  cmd = '';
  for j=1:numel(p)
    pl = plist('f', p(j).f, 'q', p(j).q);

    cmd = [cmd 'pz(' string(pl) ') '];
  end

  %%% Wrap the command only with brackets if the there are more than one object
  if numel(p) > 1
    cmd = ['[' cmd(1:end-1) ']'];
  end

  % set output
  varargout{1} = cmd;
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, 'pz', '', utils.const.categories.output, '$Id: string.m,v 1.4
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

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
%
%%%
```

```
        f = logspace(log10(f1), log10(f2), nf);
    end
  end


  %%% Now compute the response
  if Q>=0.5
    re = 1 - (f.^2./f0^2);
    im = f ./ (f0*Q);
    r = 1./complex(re, im);
  else
    re = 1;
    im = f./f0;
    r = 1./complex(re, im);
  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, 'pz', '', utils.const.categories.sigproc, '$Id: resp.m,v 1.5
2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

minfo

PURPOSE ^

MINFO a helper class for LTPDA methods.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 MINFO a helper class for LTPDA methods.
%%%
```

SUBFUNCTIONS ^

- function ii = minfo(varargin)
- function out = VEROUT()
- function ii = getInfo(varargin)
- function out = SETS()
- function out = getDefaultPlist(set)

SOURCE CODE ^

```
% 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 $
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef minfo < ltpda_nuo

  %-------------------------------------------------
  %---------- Private read-only Properties --------
  %-------------------------------------------------
  properties (SetAccess = private)
    mname      = '';
    mclass     = '';
    mpackage   = '';
    mcategory  = '';
    mversion   = '';
    sets       = {};
    plists     = [];
    argsmin    = 1;
    argsmax    = 0;
  end

  properties (SetAccess = protected)
    version = '$Id: minfo.m,v 1.15 2008/09/03 16:34:49 hewitson Exp $';
  end

  %-------------------------------------------------
  %-------- Declaration of public methods --------
  %-------------------------------------------------
  methods

    %-------------------------------------------------
    %-------- Property rules                  --------
    %-------------------------------------------------

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                                   Constructor                                %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function ii = minfo(varargin)

      %%% Call superclass
      ii = ii@ltpda_nuo(varargin{:});

      %%%%%%%%%%   Set dafault 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}.plists;
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 plists 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.plists = pls;
0135            % Check we have one set description per plist
0136            if numel(ii.plists) ~= numel(ii.sets)
0137              sets = ii.sets;
0138              while numel(sets) < numel(ii.plists)
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 plists 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];
```

```
          end
          ii.plists = pls;
          % Check we have one set description per plist
          if numel(ii.plists) ~= numel(ii.sets)
            sets = ii.sets;
            while numel(sets) < numel(ii.plists)
              sets = [sets {''}];
            end
            ii.sets = sets;
          end
          ii.argsmin = varargin{8};
        case 9
          ii.mname = varargin{1};
          ii.mclass = varargin{2};
          ii.mpackage = varargin{3};
          ii.mcategory = varargin{4};
          ii.mversion = varargin{5};
          ii.sets = varargin{6};
          % add plists until we have at least as many as set
          % descriptions
          pls = varargin{7};
          while numel(pls) < numel(ii.sets)
            pls = [pls plist];
          end
          ii.plists = pls;
          % Check we have one set description per plist
          if numel(ii.plists) ~= numel(ii.sets)
            sets = ii.sets;
            while numel(sets) < numel(ii.plists)
              sets = [sets {''}];
            end
            ii.sets = sets;
          end
          ii.argsmin = varargin{8};
          ii.argsmax = varargin{9};
        otherwise
          error('### Unknown number of constructor arguments');
      end
    end % End of constructor
  end % End public methods

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (public)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    varargout = char(varargin)
    varargout = copy(varargin)
    varargout = display(varargin)
    varargout = setMversion(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (protected)                       %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Access = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (Static, Public)                       %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true)

    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: minfo.m,v 1.15 2008/09/03 16:34:49 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'minfo');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          out = plist();
      end
```

```
    end

  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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = copy(old, deepcopy)

  if deepcopy
    ps = properties('minfo');
    s = size(old);
    obj(s(1),s(2)) = minfo;
```

```
    for kk=1:numel(old)
      utils.helper.msg(utils.const.msg.OPROC2, 'copying minfo');
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = minfo.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setMversion

PURPOSE ^

SETMVERSION Set the property 'mversion'.

SYNOPSIS ^

function varargout = setMversion(varargin)

DESCRIPTION ^

```
 SETMVERSION Set the property 'mversion'.
%%%
```

```
%
% 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
%
%%%
```

char

PURPOSE ^

CHAR convert an minfo object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert an minfo object into a string.
%%%
```

```
0017 %%%%
```

display

PURPOSE ^

DISPLAY display an minfo object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY display an minfo object.
%%%
```

```
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 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     varargout{1} = txt;
0034   elseif nargout == 0;
0035     for j=1:numel(txt)
0036       disp(txt{j});
0037     end
0038   end
0039 end
0040
0041 %%%%
```

history

PURPOSE ^

HISTORY History object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 HISTORY History object class constructor.
%%%
```

```
                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

%%%
```

```
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
depth
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 %%%%
```

```
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    %%%%
```

```
              obj = obj.a;
            case '.xml'
              root_node = xmlread(filename);
              obj = utils.helper.xmlread(root_node, 'history');
            otherwise
              error('### Unknown file type.');
          end

          %%%%%%%%%%  h = history(struct)   %%%%%%%%%%
        elseif isstruct(varargin{1})

          hstruct = varargin{1};

          obj.methodInfo   = utils.helper.struct2obj(hstruct.methodInfo, 'minfo');
          obj.plistUsed    = utils.helper.struct2obj(hstruct.plistUsed, 'plist');
          obj.methodInvars = hstruct.methodInvars;
          obj.inhists      = utils.helper.struct2obj(hstruct.inhists, 'history');
          obj.proctime     = hstruct.proctime;
          obj.version      = hstruct.version;

          %%%%%%%%%%  h = history(history)   %%%%%%%%%%
        elseif isa(varargin{1}, 'history')
          obj = copy(varargin{1}, 1);

          %%%%%%%%%%  h = history(plist)   %%%%%%%%%%
        elseif isa(varargin{1}, 'plist')
          %%% is the plist is empty then return an empty history object
          if nparams(varargin{1}) == 0
            obj.plistUsed = varargin{1};
          else
            error('### Unknown history constructor method.');
          end

        else
          error('### Unknown history constructor method.');
        end

        %%%%%%%%%%
```

```
        end
        if ~isempty(varargin{4})
          obj.methodInvars = varargin{4};
        end
        if ~isempty(varargin{5})
          obj.inhists      = varargin{5};
        end

      else
        error('### Unknown number of constructor arguments.');
      end

    end

  end


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (public)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    %%% Define Abstract methods
    varargout = char(varargin)
    varargout = copy(varargin)
    varargout = display(varargin)

    varargout = getNodes(varargin)
    varargout = getNodes_plot(varargin)
    varargout = hist2dot(varargin)
    varargout = hist2m(varargin)
    varargout = plot(varargin)
    varargout = string(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (protected)                       %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Access = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (Static, Public)                       %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true)

    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: history.m,v 1.41 2008/09/08 08:30:40 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'history');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();

        otherwise
          out = plist();
      end
    end

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (Static, Private)                      %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static = true, Access = private)
  end

end
```

*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
%%%
```

```
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = history.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
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 ^

```
% GETNODES converts a history object to a nodes structure suitable for plotting as a tree.
%%%
```

```
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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 ^

```
% GETNODES_PLOT converts a history object to a nodes structure suitable for plotting as a tree.
%%%
```

```
0081   varargout{2} = a;
0082   varargout{3} = nodes;
0083 end
0084
0085 %%%%
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a param object into a string.
%%%
```

```
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 %%%%
```

display

PURPOSE ^

DISPLAY implement terminal display for history object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY implement terminal display for history object.
%%%
```

Description of display

```
% 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
%
%%%
```

```
function plo = getDefaultPlist()
  plo = plist();
end

```

---

*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.
- getNodes 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 ^

```
% 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
%
%%%
```

```
    wstr = utils.prog.wrapstring([fcn char(node.params)], 10);
    ss = '';
    for s=wstr
      ss = [ss '\n' char(s)];
    end
    ss = ss(3:end);
    fprintf(fd, '%s_%d [%s fontsize=%d shape=%s label="%s"];\n', node.names, j, extras, fsize, shape, ss);
  end

  %%% Write node list
  fprintf(fd, '\n');
  fprintf(fd, '\n');

  for j=length(nodes):-1:1
    %   disp('----------------')
    node = nodes(j);
    if node.pn > 0
      dst  = nodes(node.pn);
      fprintf(fd, '%s_%d -> %s_%d;\n', node.names, j, dst.names, dst.n);
    else
      fprintf(fd, '%s_%d -> end;\n', node.names, j);
    end
    %   disp('----------------')
  end


  %%% close
  fprintf(fd, '}\n');

  %%% Close
  fclose(fd);

end


%%%
```

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.
- getNodes 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.
```

```
%%%
```

```
%%%
```

```
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  %%%%
```

```
0258
0259 function plo = getDefaultPlist()
0260   plo = plist();
0261 end
0262
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
% PLOT plots a history object as a tree diagram.
%%%
```

```
        plot_args{end+1} = val;
        args = args(3:end);

      else
        error('### the key [%s] is not from the type ''char''', char(args{1}));
      end
    else
      help(mfilename('fullpath'))
      error('### There is no key/value pair left.');
    end

  end

  %%% Comes the call from the browser or from the command window
  max_depth = find(pl, 'max_depth');

  %%% Output handles for figures and axes
  hfig = [];
  ax   = [];
  at   = [];

  nhists = length(hists);
  for i=1:nhists

    hist = hists(i);
    % disp(sprintf('--- plotting history %s', hist.name));

    % convert history object to a node list
    if ~isempty(max_depth)
      [n,a, nodes] = getNodes_plot(hist, [], 0, 1, [], 0, max_depth);
    else
      [n,a, nodes] = getNodes(hist, [], 0, 1, []);
    end
    p = [nodes(:).pn];
    [x,y,h]=treelayout(p);
    f = find(p~=0);
    pp = p(f);
    X = [x(f); x(pp); repmat(NaN,size(f))];
    Y = [y(f); y(pp); repmat(NaN,size(f))];
    X = X(:);
    Y = Y(:);

    % figure
    hfig = [hfig gcf];

    % axes objects
    a = plot (curr_axes, X, Y, 'r--', x, y, 'ro');
    % plot (X,Y) --> die linien
    if length(a)>1
      set(a(2), 'MarkerFaceColor', 'r');
      set(a(2), 'MarkerSize', 8);
    end
    args  = plot_args;
    while ~isempty(args)
      prop = args{1};
      val  = args{2};
      args = args(3:end);
      for ii = 1:length(a)
        set(a(ii), prop, val);
      end
    end
    ax = [ax a];

    % text objects
    a    = [];
    for j=1:length(x)
      % node description
      fcnname = getFcnName(nodes(j).names);
      str = ['{\bf\color{blue}' num2str(nodes(j).n) ':}{\bf\color{red}' fcnname '} '
char(nodes(j).params)];
      nlstr = getNodeInputs(nodes(j).invars);

      na  = text(x(j), y(j),...
        [utils.prog.wrapstring(strrep(str,'_', '\_'),getappdata(0, 'wrapstringat')) ...
        cellstr(['{\color{magenta} ' strrep(nlstr,'_', '\_') '}'])]);
      set(na, 'HorizontalAlignment', 'center');
      set(na, 'EdgeColor', 'k', 'Fontsize', 10);
      set(na, 'BackgroundColor', 'w');
      set(na, 'Margin', 5);
      a   = [a na];
    end
    at = [at a];

    % xlabel(['height = ' int2str(h)]);
    axis([0 1 0 1]);
    box off;
```

```
    axis off;

  end

  % Make outputs
  if nargout > 0
    varargout{1} = hfig;
  end
  if nargout > 1
    varargout{2} = ax;
  end
  if nargout > 2
    varargout{3} = at;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                Local Functions                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%--------------------------------------------------------------------------
% compute strings to display for function name
function fcnname = getFcnName(name)

%   switch name
%     case 'mtimes'
%       fcnname = 'x';
%     case 'times'
%       fcnname = 'x';
%     case 'plus'
%       fcnname = '+';
%     case 'minus'
%       fcnname = '-';
% %     case 'sqrt'
% %       fcnname = '\surd';
%     case 'mrdivide'
%       fcnname = '/';
%     case 'rdivide'
%       fcnname = '/';
%     otherwise
%       fcnname = char(name);
%   end

fcnname = char(name);
end

%--------------------------------------------------------------------------
% compute strings to display for inputs to nodes
function    str = getNodeInputs(invars)

  ni = length(invars);
  if ni > 0
    str = char(invars{1});
    for iv=2:ni
      s = char(invars{iv});
      if ~strcmp(s, 'pl')
        str = [str '  ' s];
      end
    end
  else
    str = '';
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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, 'history', '', utils.const.categories.output, '$Id: plot.m,v
1.27 2008/09/05 14:17:01 hewitson Exp $', sets, pl);
```

```
end

%%%
```

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 ^

```
% 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
%
%%%
```

```
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, 'history', '', utils.const.categories.output, '$Id: string.m,v 1.8 2008/09/04 15:29:30 ingo Exp $', sets, pl);
end

%%%
```

provenance

PURPOSE ^

PROVENANCE constructors for provenance class.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 PROVENANCE constructors for provenance class.
%%%
```

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 ^

```
% 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 AO 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
%
%%%
```

```
0133
0134      end % End of constructor
0135
0136    end % End public methods
0137
0138    %%%%
```

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
%%%
```

```
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = provenance.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
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.
%%%
```

```
%
%%%
```

display

PURPOSE ^

DISPLAY overload terminal display for provenance objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overload terminal display for provenance objects.
%%%
```

```
    varargout{1} = getInfo(varargin{3});
    return
  end

  % Get minfo objects
  objs = utils.helper.collect_objects(varargin(:), 'provenance');

  % get display text
  txt = utils.helper.objdisp(objs);

  % display the objects
  if nargout > 0
    varargout{1} = txt;
  elseif nargout == 0;
    for j=1:numel(txt)
      disp(txt{j});
    end
  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, 'provenance', '', utils.const.categories.output, '$Id:
display.m,v 1.12 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end

```

*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 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 ^

```
% 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
%
%%%
```

mux

PURPOSE ^

MUX concatenate params into a vector.

SYNOPSIS ^

function varargout = mux(varargin)

DESCRIPTION ^

```
 MUX concatenate params into a vector.
%%%
```

```
%
% HISTORY:      27-03-2007 M Hewitson
%                  Creation
%
%%%
```

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
%%%
```

```
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 ^

```
% 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');
```

```
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 %%%%
```

```
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
  if nargin == 1 && strcmpi(varargin{1}, 'None')
    sets = {};
    pl   = [];
  else
    sets = {'Default'};
    pl   = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'param', '', utils.const.categories.internal, '$Id: setKeyVal.m,v 1.6 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist('key', '', 'val', '');
end

```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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');
```

```
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 %%%%
```

```
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, 'param', '', utils.const.categories.internal, '$Id: setVal.m,v 1.6 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%
```

char

PURPOSE ^

CHAR convert a param object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a param object into a string.
%%%
```

```
%
%               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
%
%%%
```

```
      pstr = [pstr pp.key '=' f.name ', '];

      %%%%%%%%%%   time, timeformat, timespan   %%%%%%%%%%
    elseif isa(pp.val, 'time') || isa(pp.val, 'timeformat') || isa(pp.val, 'timespan')
      pstr = [pstr pp.key '=' char(pp.val) ', '];
      %%%%%%%%%%%%%% symbols %%%%%%%%%%%%
    elseif isa(pp.val, 'sym')
      sval = char(pp.val);
      if isempty(sval)
        sval = ['''''''];
      elseif length(sval) >= 10
        sval = [sval(1:10) '...'];
      end
      pstr = [pstr pp.key '=' sval ', '];
      %%%%%%%%%%   all other classes   %%%%%%%%%%
    else
      pstr = [pstr pp.key '=' class(pp.val) '-object, '];
    end

  end

  % remove last ', '
  if length(pstr)>1
    pstr = pstr(1:end-2);
  end

  %%% Prepare output
  varargout{1} = pstr;
end

%%%%%%%%%%%%
```

display

PURPOSE ^

DISPLAY display a parameter

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY display a parameter
%%%
```

```
%               >> 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = display(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  objs = utils.helper.collect_objects(varargin(:), 'param');

  txt = {};

  for ii = 1:numel(objs)
    banner = sprintf('---- param %d ----', i);
    txt{end+1} = banner;

    % get key and value
    name = objs(ii).key;
    v    = objs(ii).val;

    txt{end+1} = ['key: ' name];

    % convert value for printing

    %%%%%   Character   %%%%%
    if ischar(v)
      MAXLENGTH = 100;
      if length(v) > MAXLENGTH
        txt{end+1} = ['val: ''' v(1:MAXLENGTH) ' ...'''];
      else
        txt{end+1} = ['val: ''' v '''''];
      end

      %%%%%   Logical   %%%%%
    elseif islogical(v)
      if v == true
        txt{end+1} = 'val: true';
      else
        txt{end+1} = 'val: false';
      end

      %%%%%   Numeric   %%%%%
    elseif isnumeric(v)
      sval = '';
      if size(v,1) == 1 || size(v,2) == 1
        for j=1:min(length(v), 10)
          sval = [sval sprintf('%g ', v(j))];
        end
        if length(v) > 10
          sval = [sval '...'];
        end
      elseif isempty(v)
        sval = '[]';
      else
        sval = sprintf('Matrix [%dx%d]', size(v,1), size(v,2));
      end
      txt{end+1} = ['val: ' sval];

      %%%%%   Symbolic Object   %%%%%
    elseif isa(v, 'sym')
      sval = char(v);
      txt{end+1} = ['val: ' sval];

      %%%%%   Unit Object   %%%%%
    elseif isa(v, 'unit')
      sval = char(v);
      txt{end+1} = ['val: ' sval];

      %%%%%   Specwin Object   %%%%%
    elseif isa(v, 'specwin');
      txt{end+1} = ['val: specwin'];
      txt = [txt char(v)];

      %%%%%   Provenance Object   %%%%%
```

```
    elseif isa(v, 'provenance')
      txt{end+1} = ['val: ' v.creator '@' v.hostname ' (' v.ip ')'];

      %%%%%   miir and mfir Object   %%%%%
    elseif isa(v, 'miir') || isa(v, 'mfir')
      txt{end+1} = ['val: ' char(v)];

      %%%%%   time, timespan and timeformat Object   %%%%%
    elseif isa(v, 'time') || isa(v, 'timespan') || isa(v, 'timeformat')
      txt{end+1} = ['val: ' char(v)];

      %%%%%   ao                                %%%%%
      %%%%%   cdata, fsdata, tsdata, xydata   %%%%%
      %%%%%   plist, param                     %%%%%
      %%%%%   history                          %%%%%
    elseif isa(v, 'ao')     || isa(v, 'fsdata')  || isa(v, 'tsdata') || ...
        isa(v, 'cdata') || isa(v, 'xydata')  || isa(v, 'plist')  || ...
        isa(v, 'param') || isa(v, 'history')
      vinfo = whos('v');
      txt{end+1} = ['val: ' vinfo.class '-object'];

      %%%%%   pole, zero and pole/zero model Object   %%%%%
    elseif isa(v, 'pole') || isa(v, 'zero') || isa(v, 'pzmodel')
      sval = '';
      for j=1:length(v)
        sval = [sval ' ' char(v(j))];
      end
      txt{end+1} = ['val: ' sval];

      %%%%%   Structures   %%%%%
    elseif isstruct(v)

      fields = fieldnames(v);
      txt{end+1} = 'val: structure';
      for ii=1:length(fields)
        field = fields{ii};
        if ischar(v.(field))
          txt{end+1} = ['      ' field ': ''' v.(field) ''''];
        elseif isnumeric(v.(field))
          txt{end+1} = ['      ' field ': ' utils.helper.mat2str(v.(field))];
        elseif isobject(v.(field))
          txt{end+1} = ['      ' field ': ' class(v.(field)) '-object'];
        else
          txt{end+1} = ['      ' field ': ' num2str(size(v.(field),1)) 'x'
num2str(size(v.(field),2)) ' ' class(v.(field))];
        end
      end

      %%%%%   Cell   %%%%%
    elseif iscell(v)
      sval = '{';
      for ii = 1:length(v)
        if ischar(v{ii})
          sval = [sval '''' v{ii} ''' '];
        elseif isnumeric(v{ii})
          sval = [sval '[' utils.helper.mat2str(v{ii}) '] '];
        elseif isobject(v{ii})
          sval = [sval '''' class(v{ii}) ''' '];
        else
          error('### Do not use a [%s] in a cell array', class(v));
        end
      end
      if length(sval) > 1
        sval(end) = '}';
      else
        sval(end+1) = '}';
      end
      txt{end+1} = ['val: ' sval];

      %%%%%   JAVA   %%%%%
    elseif isjava(v)
      txt{end+1} = ['val: ' class(v)];

    else
      if isobject(v)
        txt{end+1} = sprintf('val: <%s>', class(v));
      elseif isa(v, 'function_handle')
        txt{end+1} = sprintf('val: function handle: %s', func2str(v));
      else
        txt{end+1} = sprintf('val: <unknown type>');
      end
    end

    banner_end(1:length(banner)) = '-';
    txt{end+1} = banner_end;

```

```
  end

  %%% Prepare output
  if nargout == 0
    for ii=1:length(txt)
      disp(txt{ii});
    end
  elseif nargout == 1
    varargout{1} = txt;
  end

end

%%%
```

display2

PURPOSE ^

DISPLAY display a parameter

SYNOPSIS ^

function varargout = display2(varargin)

DESCRIPTION ^

```
 DISPLAY display a parameter
%%%
```

```
%                  Creation
%
%%%
```

```
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 %
```

```
%   %%%%%   time, timespan and timeformat Object   %%%%%
%   elseif isa(val, 'time') || isa(val, 'timespan') || isa(val, 'timeformat')
%     txt{end+1} = [prefix char(val)];
%
%   %%%%%   ao                                %%%%%
%   %%%%%   cdata, fsdata, tsdata, xydata   %%%%%
%   %%%%%   plist, param                     %%%%%
%   %%%%%   history                           %%%%%
%   elseif isa(val, 'ao')     || isa(val, 'fsdata')  || isa(val, 'tsdata') || ...
%          isa(val, 'cdata') || isa(val, 'xydata')  || isa(val, 'plist')  || ...
%          isa(val, 'param') || isa(val, 'history')
%     vinfo = whos('val');
%     txt{end+1} = [prefix vinfo.class '-object'];

  %%%%%   JAVA   %%%%%
  elseif isjava(val)
    txt{end+1} = [prefix class(val)];

  else
    if isobject(val)
      txt{end+1} = sprintf('%s<%s>', prefix, class(val));
    else
      txt{end+1} = sprintf('%s<unknown type>', prefix);
    end
  end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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, 'param', '', utils.const.categories.output, '$Id: display2.m,v 1.4 2008/09/04 15:29:31 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist()
  plo = plist();
end
```

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.
%%%
```

```
%
% 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
%
%%%
```

```
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%
```

cdata

PURPOSE ^

CDATA is the constant data class.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 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 ^

```
% CDATA is the constant data class.
%%%
```

```
% HISTORY:  09-06-2008 Hewitson
%               Creation.
%
% SEE ALSO: ltpda_obj, ao, miir, mfir, specwin, timespan, pzmodel, history, ssm
%
%%%
```

```
          obj.y       = varargin{1}.y;
          obj.version = varargin{1}.version;

        elseif isnumeric(varargin{1})
          %%%%%%%%%%   data = cdata(data- vector, matrix)   %%%%%%%%%%
          obj = obj.setY(varargin{1});
        else
          error('### Unknown single argument constructor.');
        end

      else
        error('### Unknown number of arguments.');
      end

    end % End constructor

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (public)                                %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    varargout = char(varargin)
    varargout = copy(varargin)

    varargout = setYunits(varargin)
    varargout = setY(varargin)
    varargout = getY(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (protected)                             %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Access = protected)

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (public)                                %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (static)                                %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static)
    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: cdata.m,v 1.47 2008/09/07 18:11:30 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'cdata');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          error('### Unknown set [%s]', set');
      end
    end

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (abstract)                              %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Abstract)
  end

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

applymethod

PURPOSE ^

APPLYMETHOD applys the given method to the input 2D data.

SYNOPSIS ^

function varargout = applymethod(varargin)

DESCRIPTION ^

```
 APPLYMETHOD applys the given method to the input 2D data.
%%%
```

```
0004 % DESCRIPTION: APPLYMETHOD applys 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 %%%%
```

```
%%%
```

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.
%%%
```

```
%               >> 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
%
%%%
```

```
  ii = minfo(mfilename, 'cdata', '', utils.const.categories.internal, '$Id: applyoperator.m,v 1.8 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 pl = getDefaultPlist()
  pl = plist('op', '');
end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

char

PURPOSE ^

CHAR convert a cdata-object into a string.

SYNOPSIS ^

function varargout = char(varargin)

DESCRIPTION ^

```
 CHAR convert a cdata-object into a string.
%%%
```

```
%               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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = char(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % Collect all cdata-objects
  objs = utils.helper.collect_objects(varargin(:), '');


  %%% Add the size of the data (all)
  pstr = sprintf('(Ndata=[%sx%s],', num2str(size(objs.y,1)), num2str(size(objs.y,2)));

  %%% Add the sample rate of data (tsdata, fsdata)
  if isprop(objs, 'fs')
    pstr = sprintf('%s fs=%g,', pstr, objs.fs);
  end

  %%% Add the length of this time-series in seconds (tsdata)
  if isprop(objs, 'nsecs')
    pstr = sprintf('%s nsecs=%d,', pstr, objs.nsecs);
  end

  %%% Add number of averages (fsdata)
  if isprop(objs, 'navs')
    pstr = sprintf('%s navs=%d,', pstr, objs.navs);
  end

  %%% Add time-stamp of the first data sample (fsdata, tsdata)
  if isprop(objs, 't0')
    pstr = sprintf('%s t0=%s,', pstr, char(objs.t0));
  end

  pstr(end) = ')';

  varargout{1} = pstr;
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: char.m,v 1.9 2008/09/04 15:29:30 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:     getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
```

```
%
%%%
```

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
%%%
```

```
    obj(s(1),s(2)) = cdata;
    for kk=1:numel(old)
      utils.helper.msg(utils.const.msg.OPROC2, 'copying cdata');
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = cdata.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
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 ^

```
% 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
%
%%%
```

```
0098 end
0099
```

---

*Generated on Wed 27-Aug-2008 13:30:29 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = getY(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% get idx
  idx = [];
  if nargin == 2
    if isnumeric(varargin{2})
      idx = varargin{2};
    else
      error('### Unknown arguments to getY.');
    end
  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'.
%%%
```

```
%               >> 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
%
%%%
```

setXY

PURPOSE ^

SETXY Set the property 'xy'.

SYNOPSIS ^

function varargout = setXY(varargin)

DESCRIPTION ^

```
 SETXY Set the property 'xy'.
%%%
```

```
%               >> 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
%
%%%
```

setXunits

PURPOSE ^

SETXUNITS Set the property 'xunits'.

SYNOPSIS ^

function varargout = setXunits(varargin)

DESCRIPTION ^

```
 SETXUNITS Set the property 'xunits'.
%%%
```

```
%               >> 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
%
%%%
```

setY

PURPOSE ^

SETY Set the property 'y'.

SYNOPSIS ^

function varargout = setY(varargin)

DESCRIPTION ^

```
 SETY Set the property 'y'.
%%%
```

```
%               >> 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
%
%%%
```

setYunits

PURPOSE ^

SETYUNITS Set the property 'yunits'.

SYNOPSIS ^

function varargout = setYunits(varargin)

DESCRIPTION ^

```
 SETYUNITS Set the property 'yunits'.
%%%
```

```
%               >> 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
%
%%%
```

display

PURPOSE ^

DISPLAY implement terminal display for cdata object.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY implement terminal display for cdata object.
%%%
```

```
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 nargout == 0
0048     for ii=1:length(txt)
0049       disp(txt{ii});
0050     end
0051   end
0052
0053   varargout{1} = txt;
0054 end
0055
0056 %%%%
```

fsdata

PURPOSE ^

FSDATA frequency-series object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 FSDATA frequency-series object class constructor.
%%%
```

```
                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
```

```
%       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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef fsdata < data2D

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                            %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    t0      = time(0);
    navs    = NaN;
    fs      = NaN;
    enbw    = NaN;
    version = '$Id: fsdata.m,v 1.39 2008/09/03 16:32:18 hewitson Exp $';
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

```

```
%---------- Abstract Properties ----------
properties (Abstract = true, SetAccess = protected)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              Check property setting                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

methods
  function obj = set.t0(obj, val)
    if ~isa(val, 'time') || isempty(val)
      error('### The value for the property ''t0'' must be a time object');
    end
    obj.t0 = val;
  end
  function obj = set.navs(obj, val)
    if ~isnumeric(val) || isempty(val) || length(val) < 0 || (~isnan(val) && rem(val,1)~=0)
      error('### The value for the property ''navs'' must be a positive integer');
    end
    obj.navs = val;
  end
  function obj = set.fs(obj, val)
    if ~isempty(val)
      if ~isnumeric(val)  || ~isreal(val) || val < 0
        error('### The value for the property ''fs'' must be a real positive number');
      end
    end
    obj.fs = val;
  end
  function obj = set.enbw(obj, val)
    if ~isnumeric(val) || ~isreal(val) || any(val < 0)
      error('### The value for the property ''enbw'' must be a real positive number or a vector');
    end
    if ~isempty(val) && ~isempty(obj.y)
      if length(val) ~=1 && (length(val) ~= length(obj.y))
        error('### The ENBW can only be a single number, of a vector the same length as the y data.');
      end
    end
    obj.enbw = val;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Constructor                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

methods
  function obj = fsdata(varargin)

    %%% Call superclass
    obj = obj@data2D(varargin{:});

    %%%%%%%%%%   Set dafault values   %%%%%%%%%%

    if nargin == 0
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    elseif nargin == 1
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if isa(varargin{1}, 'fsdata')
        %%%%%%%%%%%   data = fsdata(fsdata-object)   %%%%%%%%%%
        %----------- Copy fsdata Object
        obj = copy(varargin{1}, 1);

      elseif isstruct(varargin{1})
        %%%%%%%%%%%   data = fsdata(struct)   %%%%%%%%%%
        %----------- struct
        obj.t0      = utils.helper.struct2obj(varargin{1}.t0, 'time');
        obj.navs    = varargin{1}.navs;
        obj.fs      = varargin{1}.fs;
        obj.enbw    = varargin{1}.enbw;
        obj.version = varargin{1}.version;

      elseif isnumeric(varargin{1})
        %%%%%%%%%%%   data = fsdata(y-vector)   %%%%%%%%%%
        %----------- y vector
        obj.setY(varargin{1});
```

```
          obj.setFs(1);
          obj.setX(fsdata.getFfromYFs(length(obj.y), obj.fs));

        else
          error('### Unknown single argument constructor.');
        end
      elseif nargin == 2
        %%%%
```

```
    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'fsdata');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          error('### Unknown set [%s]', set');
      end
    end

  end % End static methods

  %%%%
```

applymethod

PURPOSE ^

APPLYMETHOD applys the given method to the input 2D data.

SYNOPSIS ^

function varargout = applymethod(varargin)

DESCRIPTION ^

```
 APPLYMETHOD applys the given method to the input 2D data.
%%%
```

```
% APPLYMETHOD applys the given method to the input 2D data.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: APPLYMETHOD applys 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = applymethod(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  % Get data2D objects and plists
  ds = utils.helper.collect_objects(varargin(:), 'data2D');
  pl = utils.helper.collect_objects(varargin(:), 'plist');

  % Combine with default plist
  pl = combine(pl, getDefaultPlist);

  % Get the method to apply
  method = find(pl, 'method');
  % Get the axis we are dealing with
  axis = find(pl, 'axis');
  % Get the dimension to operate along
  dim = find(pl, 'dim');
  % Get any additional option
  opt = find(pl, 'option');

  % Loop over data objects
  for jj=1:numel(ds)
    switch upper(axis)
      case 'X'
        ds(jj).x = apply(ds(jj).getX, method, dim, opt);
      case 'Y'
        ds(jj).y = apply(ds(jj).y, method, dim, opt);
      case 'XY'
        ds(jj).x = apply(ds(jj).getX, method, dim, opt);
        ds(jj).y = apply(ds(jj).y, method, dim, opt);
      otherwise
        error('### Unknown axis to operate on.');
    end
  end

  % Set outputs
  varargout{1} = ds;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 Local Functions                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-----------------------------------------------
% Apply method to the vector v
%-----------------------------------------------
function v = apply(v, method, dim, opt)
```

```
  if ~isempty(dim) && ~isempty(opt)
    % User supplied a dimension and an option
    v = feval(method, v, dim, opt);
  elseif ~isempty(dim)
    % User supplied only a dimension
    v = feval(method, v, dim);
  elseif ~isempty(opt)
    % User supplied only an option
    v = feval(method, v, opt);
  else
    % User supplied only a method
    v = feval(method, v);
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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, 'data2D', '', utils.const.categories.internal, '$Id:
applymethod.m,v 1.4 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 pl = getDefaultPlist()
  pl = plist('method', '', 'axis', 'y', 'dim', [], 'option', '');
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%
```

```
%%%
```

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 ^

```
% 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
%
%%%
```

```
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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 ^

```
% 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
%
%%%
```

```
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:               obj.setX([1 2 3]);
0007 %               obj = obj.setX([1 2 3]); create copy of the object
0008 %
0009 % INPUTS:      obj - must be a single data2D object.
0010 %
0011 % 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:
```

```
%               >> 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
%
%%%
```

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 ^

```
% 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
%
%%%
```

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 ^

```
% 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
%
%%%
```

setY

PURPOSE ^

SETY Set the property 'y'.

SYNOPSIS ^

function varargout = setY(varargin)

DESCRIPTION ^

```
 SETY Set the property 'y'.
%%%
```

```
%               >> 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setY(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  obj = varargin{1};
  val = varargin{2};

  %%% decide whether we modify the pz-object, or create a new one.
  obj = copy(obj, nargout);

  %%% set 'y'
  obj.y = val;

  varargout{1} = obj;
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, 'data2D', '', utils.const.categories.internal, '$Id: setY.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('y', '');
end

```

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('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 ^

```
% SETYUNITS 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
%
%%%
```

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 of copy

```
%%%
```

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:               obj.setEnbw(val);
0007 %              obj = obj.setEnbw(val); create copy of the object
0008 %
0009 % INPUTS:      obj - must be a single data2D object.
0010 %
0011 % 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
%
%%%
```

setFs

PURPOSE ^

SETFS Set the property 'fs'.

SYNOPSIS ^

function varargout = setFs(varargin)

DESCRIPTION ^

```
 SETFS Set the property 'fs'.
%%%
```

```
%               >> 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
%
%%%
```

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:               obj.setNavs(val);
0007 %              obj = obj.setNavs(val); create copy of the object
0008 %
0009 % INPUTS:      obj - must be a single data2D object.
0010 %
0011 % 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
%
%%%
```

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:               obj.setT0(time(0));
0007 %              obj = obj.setT0(time(0)); create copy of the object
0008 %
0009 % INPUTS:      obj - must be a single data2D object.
0010 %
0011 % 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
%
%%%
```

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.
%%%
```

```
%
% VERSION:      $Id: char.m,v 1.4 2008/09/04 15:29:30 ingo Exp $
%
% HISTORY:      23-07-2007 Diepholz
%                  Creation
%
%%%
```

```
function plo = getDefaultPlist()
  plo = plist();
end

```

---

*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.
%%%
```

```
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 %%%%
```

```
function plo = getDefaultPlist()
  plo = plist();
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

tsdata

PURPOSE ^

TSDATA time-series object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 TSDATA time-series object class constructor.
%%%
```

```
    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 numer 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

%%%
```

- 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 ^

```
% TSDATA time-series object class constructor.
%%%
```

```
%     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 numer 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef tsdata < data2D

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                              %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    t0      = time(0);
    fs      = NaN;
    nsecs   = 0;
    version = '$Id: tsdata.m,v 1.57 2008/09/03 16:39:37 hewitson Exp $';
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %---------- Abstract Properties ----------
  properties (Abstract = true, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                             Check property setting                            %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = set.t0(obj, val)
      if (~isa(val, 'time') && ~ischar(val) && ~isnumeric(val))|| isempty(val)
        error('### The value for the property ''t0'' must be a string, a number or a time object');
      end
      if ischar(val) || isnumeric(val)
        obj.t0 = time(val);
      else
        obj.t0 = val;
      end
    end
    function obj = set.fs(obj, val)
      if ~isnumeric(val) || isempty(val) || ~isreal(val) || val < 0
        error('### The value for the property ''fs'' must be a real positive number');
      end
      obj.fs = val;
    end
    function obj = set.nsecs(obj, val)
```

```
      if ~isnumeric(val) || isempty(val) || ~isreal(val) || val < 0
        error('### The value for the property ''nsecs'' must be a real positive number');
      end
      obj.nsecs = val;
    end
  end

  %%%%
```

```
            obj.fs = 1;
            obj.t0 = varargin{2};
          else
            error('### Unknown two argument constructor.');
          end
        elseif nargin == 3
          %%%%
```

```
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    %%%%
```

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
```

```
%                  Creation
%
%%%
```

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
%%%
```

```
% HISTORY:      11-07-2008 Diepholz
%                  Creation
%
%%%
```

fixNsecs

PURPOSE ^

FIXNSECS fixes the numer of seconds.

SYNOPSIS ^

function varargout = fixNsecs(varargin)

DESCRIPTION ^

```
 FIXNSECS fixes the numer of seconds.
%%%
```

```
%%%
```

getX

PURPOSE ^

GETX Get the property 'x'.

SYNOPSIS ^

function varargout = getX(varargin)

DESCRIPTION ^

```
 GETX Get the property 'x'.
%%%
```

```
% 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
%
%%%
```

```
  ii = minfo(mfilename, 'tsdata', '', utils.const.categories.internal, '$Id: getX.m,v 1.4 2008/09/04 15:29:32 ingo Exp $', sets, pl);
end

%%%
```

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.
%%%
```

```
%%%
```

setFs

PURPOSE ^

SETFS Set the property 'fs'.

SYNOPSIS ^

function varargout = setFs(varargin)

DESCRIPTION ^

```
 SETFS Set the property 'fs'.
%%%
```

```
%               >> 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
%
%%%
```

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:               obj.setNsecs(1000);
0007 %              obj = obj.setNsecs(1000); create copy of the object
0008 %
0009 % INPUTS:      obj - must be a single data2D object.
0010 %
0011 % 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
%
%%%
```

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:               obj.setT0(time(0));
0007 %              obj = obj.setT0(time(0)); create copy of the object
0008 %
0009 % INPUTS:      obj - must be a single data2D object.
0010 %
0011 % 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
%
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for tsdata objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for tsdata objects.
%%%
```

```
%
%               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
%
%%%
```

```
  % Build info object
  ii = minfo(mfilename, 'tsdata', '', utils.const.categories.output, '$Id: display.m,v 1.18 2008/09/04 15:29:32 ingo Exp $', sets, pl);
end

%%%
```

xydata

PURPOSE ^

XYDATA X-Y data object class constructor.

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
 XYDATA X-Y data object class constructor.
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef xydata < data2D

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                Property definition                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    version = '$Id: xydata.m,v 1.32 2008/09/03 16:56:21 ingo Exp $';
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %---------- Abstract Properties ----------
  properties (Abstract = true, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Check property setting                         %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                  Constructor                                %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = xydata(varargin)

      % Call data2D constructor since we add no special features here.
      obj = obj@data2D(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%

      if nargin == 0
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      elseif nargin == 1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if isa(varargin{1}, 'xydata')
          %%%%%%%%%%%  obj = xydata(xydata-object)   %%%%%%%%%%
          obj = copy(varargin{1}, 1);

        elseif isnumeric(varargin{1})
          %%%%%%%%%%%  obj = xydata(y-vector)   %%%%%%%%%%
          obj.setY(varargin{1});
          obj.setX(1:length(varargin{1}));

        elseif isstruct(varargin{1})
          %%%%%%%%%%%   data = xydata(struct)   %%%%%%%%%%
          obj.version = varargin{1}.version;

          %%% Do nothing because all values are set in the super class

        else
          error('### Unknown single argument constructor.');
        end

      elseif nargin == 2
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%%%%  obj = xydata(x-vector, y-vector)   %%%%%%%%%%
        if numel(varargin{1}) == numel(varargin{2})
          obj.setXY(varargin{1}, varargin{2});
```

```
        end

      else
        error('### Unknown two argument constructor.');
      end
    end % End constructor
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (public)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (public)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    varargout = display(varargin);
    varargout = copy(varargin)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (protected)                       %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Access = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (static)                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Static)
    varargout = update_struct(varargin);

    function out = VEROUT()
      out = '$Id: xydata.m,v 1.32 2008/09/03 16:56:21 ingo Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'xydata');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          error('### Unknown set [%s]', set');
      end
    end

  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Methods (abstract)                        %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods (Abstract)
  end

end

```

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
%%%
```

```
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = xydata.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end

```

*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.
%%%
```

```
%
%               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
%
%%%
```

```
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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.
%%%
```

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 ^

```
% XZYDATA X-Y-Z data object class constructor.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: XZYDATA 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 y-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.
%
% 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 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef xyzdata < data3D

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                Property definition                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
    version = '$Id: xyzdata.m,v 1.12 2008/09/03 16:39:37 hewitson Exp $';
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %---------- Abstract Properties ----------
  properties (Abstract = true, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Check property setting                         %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                   Constructor                               %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = xyzdata(varargin)

      % Call data2D constructor since we add no special features here.
      obj = obj@data3D(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%

      if nargin == 0
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      elseif nargin == 1
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if isa(varargin{1}, 'xyzdata')
          %%%%%%%%%%%   xy = xydata(xydata)   %%%%%%%%%%
          obj = copy(varargin{1}, 1);

        elseif isstruct(varargin{1})
          %%%%%%%%%%%   data = xyzdata(struct)   %%%%%%%%%%
          obj.version = varargin{1}.version;

          %%% Do nothing. Because no properties are defined in this class.

        elseif isnumeric(varargin{1}) && length(varargin{1}) > 1
          %%%%%%%%%%%   xyz = xyzdata(z_vector)   %%%%%%%%%%
          obj.x = 1:size(varargin{1}, 1);
          obj.y = 1:size(varargin{1}, 2);
          obj.z = varargin{1};

        elseif isa(varargin{1}, 'plist')
          %%%%%%%%%%%  ts = xydata(plist)   %%%%%%%%%%
          if nparams(varargin{1}) == 0
            % return empty object
          else
            error('### Unknown xyzdata constructor method.');
          end

        else
          error('### Unknown xyzdata constructor method with one argument.');
        end
```

```
      elseif nargin == 3
        %%%%
```

```
  end

end

```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

getZ

PURPOSE ^

GETZ Get the property 'z'.

SYNOPSIS ^

function varargout = getZ(varargin)

DESCRIPTION ^

```
 GETZ Get the property 'z'.
%%%
```

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = getZ(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %%% get idx
  idx = [];
  if nargin == 2
    if isnumeric(varargin{2})
      idx = varargin{2};
    else
      error('### Unknown arguments to getZ.');
    end
  end

  %%% get z values
  if isempty(idx)
    varargout{1} = varargin{1}.z;
  else
    varargout{1} = varargin{1}.z(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, 'data2D', '', utils.const.categories.internal, '$Id: getZ.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

```

setZ

PURPOSE ^

SETZ Set the property 'z'.

SYNOPSIS ^

function varargout = setZ(varargin)

DESCRIPTION ^

```
 SETZ Set the property 'z'.
%%%
```

```
%               >> 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = setZ(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  obj = varargin{1};
  val = varargin{2};

  %%% decide whether we modify the pz-object, or create a new one.
  obj = copy(obj, nargout);

  %%% set 'z'
  obj.z = val;

  varargout{1} = obj;
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, 'data2D', '', utils.const.categories.internal, '$Id: setZ.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('z', '');
end
```

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 ^

```
% 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
%
%%%
```

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');
```

```
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = xyzdata.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
end

```

*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.
%%%
```

```
%
%%%
```

display

PURPOSE ^

DISPLAY overloads display functionality for xyzdata objects.

SYNOPSIS ^

function varargout = display(varargin)

DESCRIPTION ^

```
 DISPLAY overloads display functionality for xyzdata objects.
%%%
```

```
%
% 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
%
%%%
```

```
    sets = {'Default'};
    pl   = getDefaultPlist;
  end
  % Build info object
  ii = minfo(mfilename, 'xyzdata', '', utils.const.categories.output, '$Id: display.m,v 1.7 2008/09/04 15:29:32 ingo Exp $', sets, pl);
end

%%%
```

< Master index

Index for classes/@ao >

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	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.
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.
lxspec	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	TRANSPOSE 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
xspec	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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@cdata >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@data2D >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@data3D >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@fsdata >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@history >

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.
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

Subsequent directories:

- CVS

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@ltpda_data >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@ltpda_filter >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@ltpda_nuo >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@ltpda_obj >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@ltpda_uo >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@ltpda_uoh >

Index for classes/@ltpda_uoh

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%% class: ltpda_uoh %%%%%%%%%%%%%%%%%%%%%%
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'.
type	TYPE converts the input objects to MATLAB functions.

Subsequent directories:

- CVS

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@mfir >

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:

- CVS

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@miir >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@minfo >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@param >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@plist >

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 pairor 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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for classes/@provenance >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@pz >

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 comlpex 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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@pzmodel >

Index for classes/@pzmodel

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%% class: pzmodel %%%%%%%%%%%%%%%%%%%%
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	%%%%%%%%%%%%%%%%%%%

< Master index

Index for classes/@specwin >

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_FTNI	WIN_FTNI 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*

< Master index

Index for classes/@ssm >

Index for classes/@ssm

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%% class: ssm %%%%%%%%%%%%%%%%%%%%%
assemble	assembles embedded subsytems, 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	%%%%%%%%%%%%%%%%%%%%

update_struct	UPDATE_STRUCT update the input structure to the current ltpda version
validate	%%%%

< Master index

Index for classes/@time >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@timespan >

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	%%%%%%%%%%%%%%%%%%%

< Master index

Index for classes/@tsdata >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@unit >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@xydata >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for classes/@xyzdata >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/etc >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/gui/@jcontrol >

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 oveloaded for the JCONTROL class
jcontrol	JCONTROL constructor for JCONTROL class
rmappdata	RMAPPDATA function oveloaded 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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/gui/@jcontrol/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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/ao_browser >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/constructor_helper >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/gui/gltpda >

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_arrmult	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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/ltpdaRepoGUI >

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
muitable	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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/ltpdaRepoGUI/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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/ltpdalauncher >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/gui/ltpdv >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/ltpdv/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_GETWORKSPACEOBJS 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_IMPORTOBJS 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_fcnlist	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_fcnlist	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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/pzmodel_designer >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/gui/quicklook >

Index for m/gui/quicklook

Matlab files in this directory:

Contents	%%%%%%%%%%%%%%%%%%%%%% path: m/gui/quicklook %%%%%%%%%%%%%%%%%%%%%%
ltpdaquicklook	LTPDAQUICKLOOK allows the user to quicklook LTPDA objects.

Subsequent directories:

CVS

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/gui/specwin_viewer >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/helper >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index

Index for m/math >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

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 PLIST class Constructor examples of the PROVENANCE class

©LTP Team

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]
----------------
```

Constructor examples of the XYZDATA class Functions - By Category

©LTP Team

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
Contents
Contents

buildLibrary
buildquery
cat
cb_classMethodList
cb_condPlusBtn
cb_condSubtractBtn
cb_connectBtn
cb_dbSelect
cb_executeQuery
cb_fieldSelect
cb_getDBs
cb_importBtn
cb_mainfigClose
cb_refreshObjsList
cb_refreshObjsListQL
cb_retrieveBtn
cb_saveBtn
cb_serverSelect
cb_submitBtn
cb_tableSelect
cb_workspaceObjsList
cb_workspaceObjsListQL
cdata
cell_add
cell_fusion
cell_mult
cell_recut
cell_select
cellstrfind
char
char
char
char
char
char
char
char
char
char
char
char
char
char
char
close
cohere
collapseX
combine
complex
compute
computeDFT
computeInterval
computeperiodogram
conj
consolidate
conv
getFfromYFs
getFields
getNodes
getNodes_plot
getTables
getTimezones
getWorkspaceObjs
getX
getX
getY
getY
getZ
get_window
getappdata
getlowerFreq
getupperFreq
growT
gt
hist
hist2dot
hist2m
history
ifft
imag
index
interp
interpmissing
inv
iplot
isappdata
isparam
isprop
isstable
jcontrol
join
kaiser_alpha
kaiser_flatness
kaiser_nenbw
kaiser_rov
kaiser_w3db
kalman
lcohere
lcpsd
le
len
lincom
linedetect
ln
log
log10
loginLTPDA
lpsd
lt
ltf_plan
ltfe
ltpda_cohere
ltpda_constructor_helper
ltpda_constructor_helper_build
ltpda_constructor_helper_buildParams
ltpdv_workspaceList_set_data
ltpdv_zeroAdd
ltpdv_zeroClear
ltpdv_zeroDelete
lxspec
make_ltpda_objects
max
mc
mchol
md5
mdc1_ifo2acc_fd
mdc1_ifo2acc_inloop
mdc1_ifo2control
mdc1_input_noises
mdc1_x2acc
mean
median
method_fcn_template
mfir
miir
min
minfo
minreal
minus
minus
mkbandpass
mkbandpass
mkbandreject
mkbandreject
mkhighpass
mkhighpass
mklowpass
mklowpass
mlpsd_m
mlpsd_mex
mltfe
mode
modifparams
modiftimestep
modify
mpower
mpower
mrdivide
mrdivide
mtimes
mtimes
muitable
mux
ne
ngconv
nginit
ngprop
ngsetup
ngsetup_vpa
norm
nparams
pan1main
pan2param
pan4import
ssm2ss
ssmFromBuiltinSystem
ssmFromDescription
ssmFromFilename
ssmFromMiir
ssmFromPzmodel
ssmFromRepository
ssmFromss
ssm_model_delay
ssm_model_standard_system_noparams
ssm_model_standard_system_params
std
string
string
string
string
string
string
string
string
string
string
submit
subsasgn
subsref
sum
svd
t0
tan
tfe
time
timedomainfit
times
times
timeshift
timespan
tomfir
tomiir
transpose
tsdata
type
uipickfiles
uitabpanel
uitabpanel
uminus
unit
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct

conv_noisegen
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
copy
cos
cp2iir
cpsd
createTable
ctranspose
cz2iir
data2D
data3D
delay
delete
demux
det
detrend
dft
diag
diff
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display
display2
dopplercorr
ltpda_constructor_helper_class
ltpda_constructor_helper_sets
ltpda_cpsd
ltpda_data
ltpda_explorer
ltpda_filter
ltpda_gapfilling
ltpda_javacomponent
ltpda_launcher_close
ltpda_launcher_launch_ltpdagui
ltpda_launcher_launch_pzmodel
ltpda_launcher_launch_repogui
ltpda_lcohere
ltpda_lcpsd
ltpda_lincom
ltpda_linedetect
ltpda_lpsd
ltpda_ltfe
ltpda_nfest
ltpda_nuo
ltpda_obj
ltpda_polydetrend
ltpda_pwelch
ltpda_rms
ltpda_specwin_viewer_build_window
ltpda_specwin_viewer_close
ltpda_specwin_viewer_wintype
ltpda_spikecleaning
ltpda_startup
ltpda_tfe
ltpda_uitree
ltpda_uitreenode
ltpda_uo
ltpda_uoh
ltpda_user_fcn_template
ltpda_versions
ltpda_xcorr
ltpdagui
ltpdalauncher
ltpdalib
ltpdaquicklook
ltpdasim
ltpdv
ltpdv_buildFilterPanel
ltpdv_buildImportPanel
ltpdv_buildMathPanel
ltpdv_buildPlotPanel
ltpdv_buildPreprocessPanel
ltpdv_buildPropertiesPanel
ltpdv_buildSpectralPanel
ltpdv_build_buildobject
ltpdv_build_class_select
ltpdv_build_get_selected_class
ltpdv_build_get_selected_pset
ltpdv_build_params_panel
ltpdv_build_set_select
ltpdv_calc_btn
ltpdv_cb_mainfigClose
ltpdv_delete_objects
pan5output
pan6globals
pan7partial
panXyettocome
param
parseFilterParams
parseFilterParams
parse_time_string
phase
phasetrack
plist
plot
plot
plot
plus
plus
plus
polyfit
power
power
prependVersion
progressBar
provenance
psd
pset
pwelch
pz
pzm2ab
pzmodel
pzmodel_helper
rdivide
rdivide
real
redesign
redesign
reduce
reduce_model
remove
repogui
resample
resp
resp
resp
resp
resp
retrieve
ri2fq
rmappdata
rms
rp2iir
rz2iir
save
scratch
search
select
set
setA
setB
setCreated
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
update_struct
upsample
validate
validate
var
welch
welchparse
welchscale
welcome2LTPDA
win_BH92
win_Bartlett
win_FTHP
win_FTNI
win_FTSRS
win_HFT116D
win_HFT144D
win_HFT169D
win_HFT196D
win_HFT223D
win_HFT248D
win_HFT70
win_HFT90D
win_HFT95
win_Hamming
win_Hanning
win_Nuttall3
win_Nuttall3a
win_Nuttall3b
win_Nuttall4
win_Nuttall4a
win_Nuttall4b
win_Nuttall4c
win_Rectangular
win_SFT3F
win_SFT3M
win_SFT4F
win_SFT4M
win_SFT5F
win_SFT5M
win_Welch
x
xcorr
xspec
xunits
xydata
xyzdata
y
yunits
zeropad

double
downsample
ltpdv_explore_objects
ltpdv_export_objects
setCreator
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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function applymethod(as, ao_invars, method, pl, dpl, info)

  import utils.const.*

  for j=1:numel(as)
    % Message
    utils.helper.msg(msg.PROC1, 'applying %s to %s ', method, as(j).name);
    % Apply method to data
    applymethod(as(j).data, pl, plist('method', method), dpl);
    %combine the input plist and the default plist for the history
    plh = combine(pl, dpl);
    % append history
    as(j).addHistory(info, plh, ao_invars(j), as(j).hist);
    % Set new AO name
    as(j).setName([method '(' ao_invars{j} ')'], 'internal');
  end

end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

applyoperator

PURPOSE ^

APPLYOPERATOR to the analysis object

SYNOPSIS ^

function res = applyoperator(as, ao_invars, op, opsym, pl, info)

DESCRIPTION ^

```
 APPLYOPERATOR to the analysis object
%%%
```

```
% 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
%
%%%
```

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 ^

```
% 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.
%
```

```
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. Schafer, 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   nfld = floor(freqvec(:)/Fs);
0089   xm = size(xin,1); % NFFT
0090
```

```
  % Indices used by the Goertzel function (see equation 11.1 pg. 755 of [2])
  fscaled = f/Fs*xm+1;
  k = round(fscaled);

  % shift for each frequency from default xm length grid
  deltak = fscaled-k;

  tempk = k;
  % If k > xm, fold over to the 1st bin
  k(tempk > xm) = 1;
  nfld = nfld + (tempk > xm); % Make nfld reflect fold in k because of round

  n = (0:xm-1)';
  Xx = zeros(size(k,1),size(xin,2));
  for kindex = 1:length(k)
    % We need to evaluate the DFT at the requested frequency instead of a
    % neighboring frequency that lies on the grid obtained with xm number
    % of points in the 0 to fs range. We do that by giving a complex phase
    % to xin equal to the offset from the frequency to its nearest neighbor
    % on the grid. This phase translates into a shift in the DFT by the
    % same amount. The Xx(k) then is the DFT at (k+deltak).

    % apply kernal to xin so as to evaluate DFT at k+deltak)
    kernel = exp(-j*2*pi*deltak(kindex)/xm*n);
    xin_phaseshifted = xin.*repmat(kernel,1,size(xin,2));

    Xx(kindex,:) = goertzel(xin_phaseshifted,k(kindex));
  end

  % DFT computed at exactly the frequencies it was requested for
  f = freqvec(:);
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 whicch 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 ^

```
% 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 whicch 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 $
%

%   Author(s): P. Pacheco
%   Copyright 1988-2006 The MathWorks, Inc.
%   $Revision: 1.2 $  $Date: 2008/08/01 13:19:42 $

function [P,f] = computeperiodogram(x,win,nfft,esttype,varargin)

  error(nargchk(3,5,nargin,'struct'));
  if nargin < 4,
    esttype = 'psd'; % Default, compenstate for window's power.
  end

  if nargin < 5 || isempty(varargin{1}),
    Fs = 2*pi;
  else
    Fs = varargin{1};
  end

  % Validate inputs and convert row vectors to column vectors.
  [x,Lx,y,is2sig,win,errid,errmsg] = validateinputs(x,win,nfft);
  if ~isempty(errmsg), error(errid, errmsg); end

  % Determine if FFT or Goertzel
  freqVectorSpecified = false;
  if (length(nfft)> 1), freqVectorSpecified = true; end

  % Window the data
  xw = x.*win;
  if is2sig, yw = y.*win; end

  % Evaluate the window normalization constant.  A 1/N factor has been
  % omitted since it will cancel below.
  if strcmpi(esttype,'ms'),
    % The window is convolved with every power spectrum peak, therefore
    % compensate for the DC value squared to obtain correct peak heights.
    U = sum(win)^2;
  else
    U = win'*win;  % compensates for the power of the window.
  end

  % Compute the periodogram power spectrum [Power] estimate
  % A 1/N factor has been omitted since it cancels

  [Xx,f] = ao.computeDFT(xw,nfft,Fs);
  if is2sig, [Yy,f] = ao.computeDFT(yw,nfft,Fs); end

  P = Xx.*conj(Xx)/U;      % Auto spectrum.
```

```
  if is2sig,
    P = Xx.*conj(Yy)/U;  % Cross spectrum.
  end

end

%-----------------------------------------------------------------------------
function [x,Lx,y,is2sig,win,errid,errmsg] = validateinputs(x,win,nfft)
  % Validate the inputs to computexperiodogram and convert the input signal
  % vectors into column vectors.

  errid = '';
  errmsg = ''; % in case of early return.

  % Set defaults and convert to row vectors to columns.
  y     = [];
  Ly    = 0;
  is2sig= false;
  win   = win(:);
  Lw    = length(win);

  % Determine if one or two signal vectors was specified.
  Lx = length(x);
  if iscell(x),
    if length(x) > 1,
      y = x{2};
      y = y(:);
      is2sig = true;
    end
    x = x{1};
    Lx = length(x);
  end
  x = x(:);

  if is2sig,
    Ly  = length(y);
    if Lx ~= Ly,
      errid = generatemsgid('invalidInputSignalLength');
      errmsg = 'The length of the two input vectors must be equal to calculate the cross spectral density.';
      return;
    end
  end

  if Lx ~= Lw,
    errid = generatemsgid('invalidWindow');
    errmsg = 'The WINDOW must be a vector of the same length as the signal.';
    return;
  end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
given pzmodel
0002 %
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
%%%
```

*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
%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 $
%

function polzero = fq2fac(f,q)

  n = length(f);
  polzero = zeros(n,3);
  for i = 1:n
    if isnan(q(i))
      polzero(i,1:2) = [1 1/(2*pi*f(i))];
    else
      polzero(i,1:3) = [1 1/(2*pi*f(i)*q(i)) 1/((2*pi*f(i))*(2*pi*f(i)))];
    end
  end

end

```

*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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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 %s. 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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
%
% 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
%
%%%
```

```
      end
    end
    scan_format = [deblank(scan_format) '%*[^\n]'];

  %%% preallocate data array
  f_data = zeros(maxLines, read_col);

  if strcmpi(find(pl, 'Robust'), 'yes')

    f_data = robustRead(fid, f_data, columns, orig_col);

  else


    %%% Look for the first line of data
    while ~feof(fid)
      f = deblank(fgetl(fid));
      if ~isempty(f)
        if f(1) ~= comment_char
          break;
        end
      end
    end
    %%% Scan it to find how many columns we have in the file
    C = textscan(f, scan_format, 1, ...
      'CommentStyle', comment_char, ...
      'CollectOutput', 1);
    if any(isnan(C{:}))
      error('### Error in file format. Perhaps you specified more columns than the file contains?');
    end

    fseek(fid, 0, 'bof');
    %%% read file to end
    while ~feof(fid) && nlines < maxLines
      C = textscan(fid, scan_format, readlines, ...
        'CommentStyle', comment_char, ...
        'CollectOutput', 1);
      f_data(nlines+1:nlines+size(C{1},1),:) = C{1};
      nlines = nlines + length(C{1});

      if isempty(C{1})
        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);
      end

      utils.helper.msg(utils.const.msg.PROC2, 'read %09d lines of %09d', nlines, maxLines);
    end
    fclose(fid);

    %%% get only the data we want
    if size(f_data,1) > nlines
      f_data = f_data(1:nlines, :);
    end
  end


  %%%%%%%%%%   Create for each column pair the data object   %%%%%%%%%%
  if isempty(use_fs)

    %%%%%%%%%%   The numbers in columns must be straight   %%%%%%%%%%
    if mod(length(columns),2) ~= 0
      error('### the numbers in columns must be straight');
    end

    for lauf = 1:length(columns)/2

      data_x_axes = f_data(:, columns(lauf*2-1));
      data_y_axes = f_data(:, columns(lauf*2));

      % create data object corresponding to the parameter list
      ao_data = [];
      switch data_type
        case 'tsdata'
          ao_data = tsdata( data_x_axes, data_y_axes);
        case 'fsdata'
          ao_data = fsdata( data_x_axes, data_y_axes);
        case 'cdata'
          error ('### please code me up')
        case 'xydata'
          ao_data = xydata( data_x_axes, data_y_axes);
        otherwise
          error('### unknown data type ''%s''', data_type);
      end
```

```
      aa = ao(ao_data);
      aa.setName(sprintf('%s_%02d_%02d', file_name, orig_col(lauf*2-1), orig_col(lauf*2)), 'internal');
      % Add history
      pl = pl.pset('columns', [orig_col(lauf*2-1) orig_col(lauf*2)]);
      aa.addHistory(mi, pl, [], []);

      % Set the description fron the parameter list or from the file
      description_pl = find(pl, 'description');
      if ~isempty(description_pl)
        aa.description = description_pl;
      end

      a = [a aa];

    end

    %%%%%%%%%%   Create for each column AND fs a data object   %%%%%%%%%%
  else % isempty(use_fs)

    for lauf = 1:length(columns)

      data_y_axes = f_data(:, columns(lauf));

      % create data object corresponding to the parameter list
      ao_data = [];
      switch data_type
        case 'tsdata'
          ao_data = tsdata(data_y_axes, use_fs);
        case 'fsdata'
          ao_data = fsdata(data_y_axes, use_fs);
        case 'cdata'
          error ('### please code me up')
        case 'xydata'
          error ('### please code me up')
        otherwise
          error('### unknown data type ''%s''', data_type);
      end
      aa = ao(ao_data);
      aa.setName(sprintf('%s_%02d', file_name, orig_col(lauf)), 'internal');
      pl = pl.pset('columns', orig_col(lauf));
      aa.addHistory(mi, pl, [], []);

      % Set the description fron the parameter list or from the file
      description_pl = find(pl, 'description');
      if ~isempty(description_pl)
        aa.description = description_pl;
      end

      a = [a aa];

    end

  end

  %%%%%%%%%%   set fields of the AO from the parameter list   %%%%%%%%%%
  for ii = 1:length(a)

    for jj = 1:pli.nparams

      cmd = ['set' pli.params(jj).key(1) lower(pli.params(jj).key(2:end))];

      if ismethod(a(ii), cmd)
        if iscell(pli.params(jj).val) && length(pli.params(jj).val) == length(a)
          %%% Set the value in the cell
          feval(cmd, a(ii), pli.params(jj).val{ii}, 'internal');
          %%% Reset the value in the plistUset to this single value
          a(ii).hist.plistUsed.pset(pli.params(jj).key, pli.params(jj).val{ii});
        else
          %%% Set the value
          feval(cmd, a(ii), pli.params(jj).val, 'internal');
        end
      end

    end

  end % for-loop over all ao's

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                   Local Functions                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% FUNCTION:    numlines
%
% SYNTAX:      count = numlines(fid);
%
% DESCRIPTION: Number of lines in an ASCII file
%
% HISTORY:     02-08-2002 Peter Acklam, CSSM post
%                 Creation.
%
%%%
```

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
%%%
```

```
  VERSION = '$Id: fromFSfcn.m,v 1.12 2008/09/07 18:08:37 hewitson Exp $';
  % get AO info
  ii = ao.getInfo('ao', 'From Frequency-series Function');

  % Set the method version string in the minfo object
  ii.setMversion([VERSION '-->' ii.mversion]);

  % Add default values
  pl           = combine(pli, ii.plists);
  f            = find(pl, 'f');
  randn_state = find(pl, 'rand_state');

  % If the randn state is in the parameter list then set the randn function to
  % this sate
  if ~isempty(randn_state)
    randn('state',randn_state);
  else % Store the state of the random function in the parameter list
    randn_state = randn('state');
    pl.append('rand_state', randn_state);
  end

  if isempty(f)
    utils.helper.msg(msg.PROC2, 'generating f vector');
    f1 = find(pl, 'f1');
    f2 = find(pl, 'f2');
    nf = find(pl, 'nf');
    scale = find(pl, 'scale');
    switch scale
      case 'log'
        f = logspace(log10(f1), log10(f2), nf);
      case 'lin'
        f = linspace(f1, f2, nf);
      otherwise
        error('### Unknown frequency scale specified');
    end
  end

  % Get the function
  fcn = find(pl, 'fsfcn');

  % make y data
  y = eval([fcn ';']);

  fs = fsdata(f,y);
  fs.setXunits('Hz');

  % Make an analysis object
  a.data  = fs;
  % Add history
  a.addHistory(ii, pl, [], []);

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
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 sate
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:
0011 %              pl:       plist containing 'polyval', 'Nsecs', 'fs', or 't'
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
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
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.plists);
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```

  if isa(pli, 'plist')

    % Add default values
    pl = combine(pli, ii.plists);

    win = find(pl, 'win');
  elseif isa(pli, 'specwin')
    win = pli;
    pl  = plist('win', win);
  else
    error('### Unknow usage.');
  end

  % Make a cdata object
  a.data = cdata(win.win);
  a.setName(win.type, 'internal');

  % Add history
  a.addHistory(ii, pl, [], []);

end

```

*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
%%%
```

```
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.plists);
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 sate
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
0026
0027    % Combine input plist with default values
0028    pl = combine(pli, ii.plists);
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.');
0050        end
0051      end
0052      if ischar(yvals)
0053        try
0054          yvals = eval(yvals);
0055        catch
0056          error('Could not convert char array to numeric array.');
0057        end
0058      end
0059      % Make a tsdata object
0060      ts = tsdata(xvals, yvals);
0061
0062      % We have only y values
0063    elseif ~isempty(vals)
0064      if ischar(vals)
0065        try
0066          vals = eval(vals);
0067        catch
0068          error('Could not convert char array to numeric array.');
0069        end
0070      end
0071      % Make a cdata object
0072      ts = cdata(repmat(vals, 1, N));
0073
0074    else
0075      error('### Unknown constructor');
0076    end
0077
0078    % Set data
0079    a.data  = ts;
0080    % Add history
0081    a.addHistory(ii, pl, [], []);
0082
0083  end
0084
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromWaveform

PURPOSE ^

FROMWAVEFORM Construct an ao from a waveform

SYNOPSIS ^

function a = fromWaveform(a, pli)

DESCRIPTION ^

```
 FROMWAVEFORM Construct an ao from a waveform
%%%
```

```
% get AO info
ii = ao.getInfo('ao', 'From Waveform');

% Set the method version string in the minfo object
ii.setMversion([VERSION '-->' ii.mversion]);

% Add default values
pl = combine(pli, ii.plists);

nsecs  = find(pl, 'nsecs');
fs     = find(pl, 'fs');
t0     = find(pl, 't0');
if isempty(t0)
  t0 = time(0);
end

waveform = lower(find(pl, 'waveform'));
switch waveform
  %------------ Sine Wave
  case {'sine wave','sinewave','sine-wave'}
    ampl = find(pl, 'A');
    freq = find(pl, 'f');
    phi  = find(pl, 'phi');

    if length(ampl) ~= length(freq) || length(freq) ~= length(phi) || length(ampl) ~= length(phi)
      error('### Provide an Amplitude, frequency and phase for each sine-wave in the series.');
    end

    if isempty(ampl)
      ampl = 1;
    end
    if isempty(freq)
      freq = 1;
    end
    if isempty(phi)
      phi = 0;
    end

    % build command
    tsfcn = '';
    for jj=1:length(ampl)
      amp = ampl(jj);
      fr = freq(jj);
      ph = phi(jj);

      if jj==1
        tsfcn = sprintf('%g*sin(2*pi*%g*t + %g*pi/180)', amp, fr, ph);
      else
        tsfcn = [tsfcn sprintf(' + %g*sin(2*pi*%g*t + %g*pi/180)', amp, fr, ph)];
      end
    end

    %------------ Noise
  case 'noise'
    ntype = find(pl, 'type');
    if isempty(ntype)
      ntype = 'Normal';
    end

    randn_state = find(pl, 'rand_state');

    switch lower(ntype)
      case 'normal'
        tsfcn = 'randn(size(t))';
        % Set random state
        if ~isempty(randn_state)
          randn('state',randn_state);
        else
          randn_state = randn('state');
          pl.append('rand_state', randn_state);
        end

      case 'uniform'
        tsfcn = 'rand(size(t))';
        % Set random state
        if ~isempty(randn_state)
          rand('seed',randn_state);
        else
          randn_state = rand('seed');
          pl.append('rand_state', randn_state);
        end

    end
    %------------ Chirp
```

```
    case 'chirp'
      f0  = find(pl, 'f0');
      fe  = find(pl, 'f1');
      te  = find(pl, 't1');
      if isempty(f0)
        f0 = 0;
      end
      if isempty(fe)
        fe = fs/2;
      end
      if isempty(te)
        te = nsecs;
      end
      tsfcn = sprintf('chirp(t,%g,%g,%g)', f0, fe, te);
      %------------ Gaussian pulse
    case {'gaussian pulse','gaussian-pulse'}
      fc  = find(pl, 'f0');
      bw  = find(pl, 'bw');
      if isempty(fc)
        fc = 1;
      end
      if isempty(bw)
        bw = fs/2;
      end
      tsfcn = sprintf('gauspuls(t,%g,%g)', fc, bw);
    case {'square wave','squarewave','square-wave'}
      freq = find(pl, 'f');
      duty = find(pl, 'duty');
      if isempty(freq)
        freq = 1;
      end
      if isempty(duty)
        duty = 50;
      end
      tsfcn = sprintf('square(2*pi*%g*t,%g)', freq, duty);
    case {'saw tooth','sawtooth','saw-tooth'}
      freq  = find(pl, 'f');
      width = find(pl, 'width');
      if isempty(freq)
        freq = 1;
      end
      if isempty(width)
        width = 0.5;
      end
      tsfcn = sprintf('sawtooth(2*pi*%g*t,%g)', freq, width);
  end


  % construct tsdata
  t = linspace(0, nsecs-1/fs, nsecs*fs);
  % make y data
  y = eval([tsfcn ';']);

  ts = tsdata(t,y);
  ts.setXunits('s');
  ts.setYunits('V');
  ts.setT0(t0);

  % Make an analysis object
  a.data  = ts;
  a.setName(waveform, 'internal');
  % Add history
  a.addHistory(ii, pl, [], []);

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
- xspec XSPEC performs cross-spectral analysis of various forms.

SUBFUNCTIONS ^

- function ii = getInfo(varargin)
- function plo = getDefaultPlist()

SOURCE CODE ^

```
% FS Get the data property 'fs'.
%%%
```

```
0048 %
0049 % DESCRIPTION: Get Info Object
0050 %
0051 % HISTORY:     11-07-07 M Hewitson
0052 %                 Creation.
0053 %
0054 %%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%       q       - a vector of start indices for the segments

function varargout = ltf_plan(varargin)

  % Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  %% ------ Check inputs     ------------------------------------------------
  if nargin ~= 7 || nargout ~= 5
    help(mfilename)
    error('### Incorrect usage');
  end

  Ndata  = varargin{1};
  fs     = varargin{2};
  olap   = varargin{3};
  bmin   = varargin{4};
  Lmin   = varargin{5};
  Jdes   = varargin{6};
  Kdes   = varargin{7};

  %% ------ Set up some variables ------------------------------------------

  xov     = (1 - olap/100);
  fmin    = fs / Ndata * bmin;
  fmax    = fs/2;
  fresmin = fs / Ndata;
  freslim = fresmin * (1+xov*(Kdes-1));
```

```
  logfact = (Ndata/2)^(1/Jdes) - 1;



  %% ------ Prepare outputs       ------------------------------------------

  f = [];
  r = [];
  b = [];
  L = [];
  K = [];
  % q = [];

  %% ------ Loop over frequency   ------------------------------------------
  fi = fmin;
  while fi < fmax

    fres = fi * logfact;
    if fres <= freslim
      fres = sqrt(fres*freslim);
    end
    if fres < fresmin
      fres = fresmin;
    end

    bin = fi/fres;
    if bin < bmin
      bin = bmin;
      fres = fi/bin;
    end

    dftlen = round(fs / fres);
    if dftlen > Ndata
      dftlen = Ndata;
    end
    if dftlen < Lmin
      dftlen = Lmin;
    end

    nseg = round((Ndata - dftlen) / (xov*dftlen) + 1);
    if nseg == 1
      dftlen = Ndata;
    end

    fres = fs / dftlen;
    bin  = fi / fres;

    % Store outputs
    f = [f fi];
    r = [r fres];
    b = [b bin];
    L = [L dftlen];
    K = [K nseg];

    fi = fi + fres;

  end

  %% ------ Set outputs           ------------------------------------------

  varargout{1} = f.';
  varargout{2} = r.';
  varargout{3} = b.';
  varargout{4} = L.';
  varargout{5} = K.';
end

%-----------------------------------------------------------------------
% Get Info Object
%-----------------------------------------------------------------------
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, 'ao', '', utils.const.categories.relop, '$Id: ltf_plan.m,v 1.4
2008/09/05 11:05:29 ingo Exp $', sets, pl);
end

%-----------------------------------------------------------------------
% Get Default Plist
%-----------------------------------------------------------------------
```

```
function plo = getDefaultPlist()

  plo = plist(...
    'Ndata', 10, ...
    'fs',    10, ...
    'olap',  50, ...
    'bmin',  1, ...
    'Lmin',  1e20, ...
    'Jdes',  1000, ...
    'Kdes',  100 ...
    );
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

lxspec

PURPOSE ^

LXSPEC performs log-scale cross-spectral analysis of various forms.

SYNOPSIS ^

function varargout = lxspec(varargin)

DESCRIPTION ^

```
 LXSPEC performs log-scale cross-spectral analysis of various forms.
%%%
```

```
%%%
```

```
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   Nolap = find(pl, 'Olap');  % desired overlap
0101   if Nolap == -1
0102     % use window function rov
0103     Nolap = Win.rov/100;
0104     utils.helper.msg(msg.PROC1, 'using recommended overlap of %d', Nolap);
0105     pl = pset(pl, 'Olap', Nolap);
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, Nolap, 1, Lmin, Jdes, Kdes);
0112
0113   %----------------- compute TF Estimates
0114   Txy = ao.mltfe(iS, f, r, m, L, fsmax, Win, Order, Nolap*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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% MCHOL
function varargout = mchol(m)

  leng = length(m);
  for i=1:leng
    dets(i) = det(m(1:i, 1:i));
  end

  sqdet = sqrt(dets);
  if isreal('sqdet')==0
    error('### matrix not positive definite')
  end

  if(isa(m,'sym'))
    M = m;
    n = length( M );
    L = vpa(zeros( n, n ));
    for i=1:n
      L(i, i) = sqrt( M(i, i) - L(i, :)*L(i, :)' );
      for j=(i + 1):n
        L(j, i) = ( M(j, i) - L(i, :)*L(j, :)' )/L(i, i);
      end
    end
    % L = solve('L');
    varargout{1} = L;
  end

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      %%%%
```

```
      f2    = find(pl, 'f2');
      nf    = find(pl, 'nf');
      scale = find(pl, 'scale');

      switch scale
        case 'lin'
          f   = linspace(f1, f2, nf);
        case 'log'
          f = logspace(log10(f1), log10(f2), nf);
      end
    end

    source = find(pl, 'source');

    %% Compute response for frequencies f

    switch source
      case 'ifo'
        a = getIfoNoise(f);
      case 'tm'
        a = getTMNoise(f);
      case 'sc'
        a = getSCNoise(f);
      otherwise
        error('### Unknown noise source requested.');
    end

    varargout{1} = a;
    end
    %--------------------------------------------------------------------------
    % Get IFO noise for each frequency
    function o = getIfoNoise(f)

      % Description of the noise
      fcn = '0.5*(5e-12)^2 * ( 1 + (1+1e8)./(1+f.^2/1e-12) .* (1 + (1+9e6)./(1+f.^2/1e-14) ))';
      % Pack in a plist
      pl  = plist('fsfcn', fcn, 'f', f);

      % Build AO from plist
      o = ao(pl);
      o.setName('o_nxx', 'internal');
      o.setXunits('Hz', 'internal');
      o.setYunits('m^2/Hz', 'internal');
    end
    %--------------------------------------------------------------------------
    % Get SC noise for each frequency
    function o = getSCNoise(f)

      % Zeros as constant factors
      z1 = 1 + (10e-3/1e-6)^2;
      z2 = 1 + (0.1e-3/0.1e-6)^2;
      % AOs for frequency-dependent parts
      p1 = ao(plist('fsfcn', '1 + (f./100).^2', 'f', f));
      p2 = ao(plist('fsfcn', '1 + (f/1e-6).^2', 'f', f));
      p3 = ao(plist('fsfcn', '1 + (f/0.1e-6).^2', 'f', f));

      % Combine the parts
      G = (0.1e-6/436)^2;
      o = G .* (1./p1 + (z1./p2).*(1 + z2./p3 ));

      % Set properties of final AO
      o.setName('A_nxx', 'internal');
      o.setXunits('Hz', 'internal');
      o.setYunits('m^2s^{-4}/ Hz', 'internal');

    end
    %--------------------------------------------------------------------------
    % Get TM noise for each frequency
    function o = getTMNoise(f)

      % Construct the various parts of the spectrum

      % zeros
      z1 = ao(plist('fsfcn', '(1 + (f./100).^2)', 'f', f));
      z2 = (1 + (3e-3/1e-6).^2);
      z3 = ao(plist('fsfcn', '(1 + (f./1e-6).^2)', 'f', f));
      z4 = (1 + (0.1e-3./0.1e-6).^2);
      z5 = ao(plist('fsfcn', '(1 + (f./0.1e-6).^2)', 'f', f));

      % poles
      p1 = (1 + (1e-3./100).^2);
      p2 = (1 + (3e-3./1e-6).^2);
      p3 = (1 + (1e-3./1e-6).^2);
      p4 = (1 + (0.1e-3./0.1e-6).^2);
      p5 = (1 + (1e-3./0.1e-6).^2);
```

```
      % Combine the parts
      o = (30e-15)^2 .* ( (1./z1 + z2./z3.*(1 + z4./z5))  ./ (1./p1 + p2./p3.*(1 + p4./p5)));

      % Set properties of final AO
      o.setName('A_1xx', 'internal');
      o.setXunits('Hz', 'internal');
      o.setYunits('m^2s^{-4}/ Hz', 'internal');

    end
    %% Default parameters
    function plo = getDefaultPlist(varargin)
      % List of available parameter sets
      sets = {'List', 'Range'};
      if nargin == 0
        plo = sets;
        return
      end
      set = varargin{1};
      switch set
        case 'List'
          plo = plist('source', 'ifo', 'f', [1]);
        case 'Range'
          plo = plist('source', 'ifo', ...
                      'f1', 1e-6,...
                      'f2', 5,...
                      'nf', 1000,...
                      'scale', 'log');
        otherwise
          plo = plist();
        end
    end

    %-------------------------------------------------------------------------
    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, 'ao', '', utils.const.categories.mdc01, '$Id: mdc1_input_noises.m,v 1.3 2008/09/05 11:05:29 ingo Exp $', sets, pl);
    end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.psll);
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
```

```
  end % for j=1:nf

  varargout{1} = S;
  varargout{2} = Sxx;
  varargout{3} = ENBW;

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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);
```

```
  S    = zeros(nf,1);
  ENBW = zeros(nf,1);

  disp_each = round(nf/100)*10;

  minReached = 0;

  for j=1:nf

    if mod(j,disp_each) == 0 || j == 1 || j == nf
      utils.helper.msg(msg.PROC1, 'computing frequency %04d of %d: %f Hz', j, nf, f(j));
    end

    % compute DFT exponent and window
    l = L(j);

    % Check if we need to update the window values
    % - once we reach Lmin, the window values don't change.
    if ~minReached
      switch win.type
        case 'Kaiser'
          win = specwin(win.type, l, win.psll);
        otherwise
          win = specwin(win.type, l);
      end
      if l==Lmin
        minReached = 1;
      end
    end

    p     = 1i * twopi * m(j)/l.*(0:l-1);
    C     = win.win .* exp(p);

    % Core DFT part in C-mex file
    [A, nsegs] = ltpda_dft(x, l, C, olap, order);

    if mod(j,disp_each) == 0 || j == 1 || j == nf
      utils.helper.msg(msg.PROC2, 'averaged %d segments', nsegs);
    end

    A2ns    = 2.0*A/nsegs;
    S1      = win.ws;
    S12     = S1*S1;
    S2      = win.ws2;
    ENBW(j) = fs*S2/S12;
    Sxx(j)  = A2ns/fs/S2;
    S(j)    = A2ns/S12;

  end

  varargout{1} = S;
  varargout{2} = Sxx;
  varargout{3} = ENBW;

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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;
```

```
minReached = 0;

% ----- Loop over Frequency
for fi=1:nf

  % compute DFT exponent and window
  l = L(fi);

  % Check if we need to update the window values
  % - once we reach Lmin, the window values don't change.
  if ~minReached
    switch win.type
      case 'Kaiser'
        win = specwin(win.type, l, win.psll);
      otherwise
        win = specwin(win.type, l);
    end
    if l==Lmin
      minReached = 1;
    end
  end

  % Compute DFT coefficients
  p     = 1i * twopi * m(fi)/l.*[0:l-1];
  C     = win.win .* exp(p);

  if mod(fi,disp_each) == 0 || fi == 1 || fi == nf
    utils.helper.msg(msg.PROC1, 'computing frequency %04d of %04d: %f Hz', fi, nf, f(fi));
  end

  % Loop over input channels
  for j=1:nc
    % loop over output channels
    for k=1:nc

      switch method
        case 'tfe'
          if k~=j
            % Core cross-DFT part in C-mex file
            % We need cross-spectrum and Power spectrum
            [XY, XX, YY, nsegs] = ltpda_dft(X(j,:), X(k,:), l, C, olap, order);
            Txy(j,k,fi)  = XY/XX;
          else
            % Trivial case of equal input and output
            Txy(j,k,fi)  = 1;
          end

        case 'cpsd'
          if k~=j
            % Core cross-DFT part in C-mex file
            [A, XX, YY, nsegs] = ltpda_dft(X(j,:), X(k,:), l, C, olap, order);
          else
            % Core DFT part in C-mex file
            [A, nsegs] = ltpda_dft(X(j,:), l, C, olap, order);
          end
          S1      = win.ws;
          S2      = win.ws2;
          Txy(j,k,fi)  = 2.0*A/nsegs/fs/S2;

        case 'cohere'

          if k>j
            % Core cross-DFT part in C-mex file
            % We need cross-spectrum and Power spectrum
            [XY, XX, YY, nsegs] = ltpda_dft(X(j,:), X(k,:), l, C, olap, order);
            Txy(j,k,fi)  = (abs(XY).^2)/(XX.*YY);
          else
            % Trivial case of equal input and output
            Txy(j,k,fi)  = 1;
          end
      end
    end % End loop over output channels
  end % End loop over input channels
end % End loop over frequencies

switch method
  case 'cohere'
    % Fill other elements
    for j=1:nc
      for k=1:nc
        if k<j
          Txy(j,k,:) = Txy(k,j,:);
        end
      end
    end
```

```
    case 'tfe'
    case 'cpsd'
  end

  % Set output
  varargout{1} = Txy;

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 $

function [a,b] = ngconv(pzm)

  zs = pzm.zeros;
  ps = pzm.poles;

  f_zer = zeros(length(zs));
  q_zer = zeros(length(zs));
  for j=1:length(zs)
    z = zs(j);
    f_zer(j,1) = z.f;
    q_zer(j,1) = z.q;
    %if isnan(q_zer(j))
    %    q_zer(j,1) = 0;
    %end
    %zv(j,1:2) = [f q];
  end

  f_pol = zeros(length(ps));
  q_pol = zeros(length(ps));
  for j=1:length(ps)
    p = ps(j);
    f_pol(j,1) = p.f;
    q_pol(j,1) = p.q;
```

```
    %if isnan(q_pol(j))
    %    q_pol(j,1) = 0;
    %end
  end

  %%% calculate factors from f and q
  pol = ao.fq2fac(f_pol,q_pol);
  zer = ao.fq2fac(f_zer,q_zer);

  %%%
  [b,a] = ao.conv_noisegen(pol,zer);
end


```

*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 ^

```
% 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 $

function y = nginit(Tinit)

  n = length(Tinit);

  % writing the generator
  r = randn(n,1);
  y = Tinit * r;

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %
```

```
% $Id: ngprop.m,v 1.2 2008/08/01 13:19:42 ingo Exp $

function [x y] = ngprop(Tprop, E, num, y, ns)

  lengT = length(Tprop);
  lengb = lengT+1;

  num=num';
  num = [num zeros(1,(lengb-length(num)-1))];


  x = zeros(ns,1);
  for i=1:ns
    r = randn(lengT,1);
    y = E * y + Tprop * r;
    x(i) = num*y;
  end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
```

```
n = length(den)-1;

%% setting up matrix Aij

m_a = zeros(n,n);
for i = 1:n
  for j = 1:n
    if j == i+1
      m_a(i,j) = 1;
    end
    if i == n
      m_a(i,j) = -den(j);
    end
  end
end

%% Matrix exponential E
E = expm(m_a*dt);

%% setting up matrix Bij
B = zeros(n,n);
for i=1:n
  if rem(i,2) ~= 0
    j0 = (i+1)/2;
    s  = (-1)^(j0+1);
    j  = j0;
    for k=1:2:(n+1)
      B(i,j) = s*den(k);
      s    = -s;
      j = j+1;
    end
  end
  if rem(i,2) == 0
    j0 = i/2+1;
    s  = (-1)^j0;
    j  = j0;
    for k=2:2:(n+1)
      B(i,j) = s * den(k);
      s         = -s;
      j         = j+1;
    end
  end
end

%% solve B * m = k
m_k = zeros(n,1);
m_k(n) = 0.5;
m_m = B\m_k;

%% filling covariance matrix Cinit
Cinit = zeros(n,n);
for i=1:n
  for j=1:n
    if rem((i+j),2) == 0    % even
      Cinit(i,j) = (-1)^((i-j)/2) * m_m((i+j)/2);
    else
      Cinit(i,j) = 0;
    end
  end
end


%cholesky decomposition

Tinit = chol(Cinit,'lower'); %lower triangular matrix

%% setting up matrix D
N = n*(n+1)/2;

m_d = zeros(N);
g = zeros(n);
for i=1:n
  for j=1:n
    if i>=j
      g(i,j) = (i*i-i)/2+(j);
    else
      g(i,j) = (j*j-j)/2+(i);
    end
  end
end

for i=1:n
  for j=i:n
    for k=1:n
      m_d(g(i,j),g(j,k)) = m_d(g(i,j),g(j,k)) + m_a(i,k);
      m_d(g(i,j),g(i,k)) = m_d(g(i,j),g(i,k)) + m_a(j,k);
```

```
      end
    end
  end


  %% setting up q from D * p = q
  m_q = zeros(1,g(n,n));
  for i=1:n
    for j=i:n
      if i==n
        m_q(g(i,j)) = (E(n,n))*(E(n,n))-1;
      else
        m_q(g(i,j)) = (E(i,n)*E(j,n));
      end
    end
  end

  m_p = m_d\m_q';

  Cprop = zeros(n);
  for i=1:n
    for j=1:n
      Cprop(i,j) = m_p(g(i,j));
    end
  end
  Tprop = chol(Cprop,'lower');

  % %% writing the generator
  % r = randn(n,1);
  % y = Tinit * r;
  % x = zeros(Nfft,1);
  % for i=1:Nfft
  %     r = randn(n,1);
  %     y = E * y + Tprop * r;
  %     x(i) = a*y;
  % end

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function [Tinit,Tprop,E] = ngsetup_vpa(den,fs, ndigits)

  % ALGONAME = mfilename;
  % VERSION  = '$Id: ngsetup_vpa.m,v 1.2 2008/08/01 13:19:42 ingo Exp $';
  digits(ndigits)
  fs = sym(fs);
  den = sym(den,'d');
  den=den';
  dt = vpa(1/fs);

  n = length(den)-1;
  % digits(d);

  %% setting up matrix Aij
  m_a = vpa(zeros(n,n));
  for i = 1:n
    for j = 1:n
      if j == i+1
        m_a(i,j) = 1;
      end
      if i == n
        m_a(i,j) = -den(j);
      end
    end
  end

  %% Matrix exponential E
  a = m_a*dt;
  E = expm(a);


  %% setting up matrix Bij
  B = vpa(zeros(n,n));
  for i=1:n
    if rem(i,2) ~= 0
      j0 = (i+1)/2;
      s  = (-1)^(j0+1);
      j  = j0;
      for k=1:2:(n+1)
```

```
        d1 = den(k);
        d2 = s*d1;
        B(i,j) = d2;
        s   = -s;
        j = j+1;
      end
    end
    if rem(i,2) == 0
      j0 = i/2+1;
      s  = (-1)^j0;
      j  = j0;
      for k=2:2:(n+1)
        d1 = den(k);
        d2 = s*d1;
        B(i,j) = d2;
        s        = -s;
        j        = j+1;
      end
    end
  end

  %% solve B * m = k
  m_k = vpa(zeros(n,1));
  m_k(n) = 0.5;
  m_m = vpa(B\m_k);

  %% filling covariance matrix Cinit
  % Cinit = vpa(zeros(n,n));
  % for i=1:n
  %     for j=1:n
  %         if rem((i+j),2) == 0    % even
  %             d1 = (-1)^((i-j)/2);
  %             d1 = subs(d1)
  %             d2 = vpa('m_m((i+j)/2)');
  %             d2 = subs(d2)
  %             d3 = vpa(ctranspose(d2));
  %             d3 = subs(d3)
  %             d4 = vpa('d1 * d3');
  %             d4 = subs(d4)
  %             Cinit(i,j) = d4;
  %         else
  %             Cinit(i,j) = 0;
  %         end
  %     end
  % end

  Cinit = vpa(zeros(n,n));
  for i=1:n
    for j=1:n
      if rem((i+j),2) == 0    % even
        d1 = (-1)^((i-j)/2);
        d2 = (i+j)/2;
        Cinit(i,j) = d1 * m_m(d2);
        %         else
        %             Cinit(i,j) = 0;
      end
    end
  end
  %cholesky decomposition

  % Tinit = chol(double(Cinit),'lower'); %lower triangular matrix
  Tinit = ao.mchol(Cinit);

  %% setting up matrix D
  N = n*(n+1)/2;

  m_d = vpa(zeros(N));
  g = zeros(n);
  for i=1:n
    for j=1:n
      if i>=j
        g(i,j) = (i*i-i)/2+(j);
      else
        g(i,j) = (j*j-j)/2+(i);
      end
    end
  end

  for i=1:n
    for j=i:n
      for k=1:n
        m_d(g(i,j),g(j,k)) = m_d(g(i,j),g(j,k)) + m_a(i,k);
        m_d(g(i,j),g(i,k)) = m_d(g(i,j),g(i,k)) + m_a(j,k);
      end
    end
  end
```

```


  %% setting up q from D * p = q
  m_q = vpa(zeros(1,g(n,n)));
  for i=1:n
    for j=i:n
      if i==n
        m_q(g(i,j)) = (E(n,n))*(E(n,n))-1;
      else
        m_q(g(i,j)) = (E(i,n)*E(j,n));
      end
    end
  end

  m_p = m_d\m_q';

  Cprop = vpa(zeros(n));
  for i=1:n
    for j=1:n
      Cprop(i,j) = m_p(g(i,j));
    end
  end

  Tprop = ao.mchol(Cprop);
  Tprop = double(Tprop);
  E = double(E);
  Tinit = double(Tinit);

  % Tprop = chol(Cprop,'lower');

  % Tprop = chol(double((Cprop)),'lower');
  % Tprop = mchol(Cprop);

  % %% writing the generator
  % r = randn(n,1);
  % y = Tinit * r;
  % x = zeros(Nfft,1);
  % for i=1:Nfft
  %     r = randn(n,1);
  %     y = E * y + Tprop * r;
  %     x(i) = a*y;
  % end

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = t0(varargin)

  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  obj = varargin{1};

  if isprop(obj.data, 't0')
    varargout{1} = obj.data.t0;
  else
    varargout{1} = [];
  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, 'ao', '', utils.const.categories.internal, '$Id: t0.m,v 1.1
2008/09/05 13:55:17 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
function plo = getDefaultPlist()
  plo = plist();
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
%%%
```

```
  %%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

  if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
      strcmp(struct_ver, '1.0')

    try

      obj_struct.creator = obj_struct.provenance;
      obj_struct = rmfield(obj_struct, 'provenance');

      obj_struct = rmfield(obj_struct, 'plist');

      obj_struct.procinfo = plist();
      obj_struct.plotinfo = plist();

    catch ME
      disp(varargin{1});
      throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (ao) above is not from version 1.0')));
    end

    % Set the struct_ver to '1.1'
    struct_ver = '1.1';

  end

  %%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

  if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
      strcmp(struct_ver, '1.1')

    % Set the struct_ver to '1.2'
    struct_ver = '1.1';

  end

  %%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

  if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
      strcmp(struct_ver, '1.2')

    % Set the struct_ver to '1.3'
    struct_ver = '1.3';

  end

  varargout{1} = obj_struct;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 %
```

```
% [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 $
%

%   Author(s): P. Pacheco
%   Copyright 1988-2006 The MathWorks, Inc.
%   $Revision: 1.5 $  $Date: 2008/09/03 17:05:51 $

%   References:
%     [1] Petre Stoica and Randolph Moses, Introduction To Spectral
%         Analysis, Prentice-Hall, 1997, pg. 15
%     [2] Monson Hayes, Statistical Digital Signal Processing and
%         Modeling, John Wiley & Sons, 1996.

% Compute the periodogram power spectrum of each segment and average always
% compute the whole power spectrum, we force Fs = 1 to get a PS not a PSD.

function varargout = welch(varargin)

  if nargin == 3
    a  = varargin{1};
    esttype = varargin{2};
    pl = varargin{3};
    x  = a.data.y;
    inunits = a.data.yunits;
  else
    a  = varargin{1};
    b  = varargin{2};
    esttype = varargin{3};
    pl = varargin{4};
    if a.data.fs ~= b.data.fs
      error('### Two time-series have different sample rates.');
    end
    inunits = b.data.yunits / a.data.yunits;
    x  = {a.data.y, b.data.y};
  end

  % Parse inputs
  win     = find(pl, 'Win');
  nfft    = find(pl, 'Nfft');
  olap    = find(pl, 'Olap')/100;
  scale   = find(pl, 'scale');
  Xolap   = floor(olap*nfft);
  fs      = a.data.fs;
  order   = find(pl, 'order');

  [x,M,isreal_x,y,Ly,win,winName,winParam,noverlap,k,L,options] = ...
    ao.welchparse(x,esttype,win.win, Xolap, nfft, fs);

  % Initialize
  Sxx = zeros(options.nfft,1);

  % Frequency vector was specified, return and plot two-sided PSD
  freqVectorSpecified = false; nrow = 1;
  if length(options.nfft) > 1,
    freqVectorSpecified = true;
    [ncol,nrow] = size(options.nfft);
  end

```

```
% Compute the periodogram power spectrum of each segment and average always
% compute the whole power spectrum, we force Fs = 1 to get a PS not a PSD.

% Initialize
if freqVectorSpecified,
  Sxx = zeros(length(options.nfft),1);
else
  Sxx = zeros(options.nfft,1);
end
range = options.range;

LminusOverlap = L-noverlap;
xStart = 1:LminusOverlap:k*LminusOverlap;
xEnd   = xStart+L-1;
switch esttype,
  case {'ms','psd'},
    for i = 1:k,
      if order < 0
        seg = x(xStart(i):xEnd(i));
      else
        [seg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
      end
      [Sxxk,w] = ao.computeperiodogram(seg,win,...
        options.nfft,esttype,options.Fs);
      Sxx  = Sxx + Sxxk;
    end
  case 'cpsd',
    for i = 1:k,
      if order < 0
        xseg = x(xStart(i):xEnd(i));
      else
        [xseg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
      end
      if order < 0
        yseg = y(xStart(i):xEnd(i));
      else
        [yseg,coeffs] = ltpda_polyreg(y(xStart(i):xEnd(i)), order);
      end
      [Sxxk,w] =  ao.computeperiodogram({xseg,...
        yseg},win,options.nfft,esttype,options.Fs);
      Sxx  = Sxx + Sxxk;
    end
  case 'tfe'
    Sxy = zeros(options.nfft,1); % Initialize
    for i = 1:k,
      if order < 0
        xseg = x(xStart(i):xEnd(i));
      else
        [xseg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
      end
      if order < 0
        yseg = y(xStart(i):xEnd(i));
      else
        [yseg,coeffs] = ltpda_polyreg(y(xStart(i):xEnd(i)), order);
      end
      [Sxxk,w] = ao.computeperiodogram(xseg,...
        win,options.nfft,esttype,options.Fs);
      [Syxk,w] = ao.computeperiodogram({yseg,...
        x(xStart(i):xEnd(i))},win,options.nfft,esttype,options.Fs);
      Sxx  = Sxx + Sxxk;
      Sxy  = Sxy + Syxk;
    end
  case 'mscohere'
    % Note: (Sxy1+Sxy2)/(Sxx1+Sxx2) != (Sxy1/Sxy2) + (Sxx1/Sxx2)
    % ie, we can't push the computation of Cxy into computeperiodogram.
    Sxy = zeros(options.nfft,1); % Initialize
    Syy = zeros(options.nfft,1); % Initialize
    for i = 1:k,
      if order < 0
        xseg = x(xStart(i):xEnd(i));
      else
        [xseg,coeffs] = ltpda_polyreg(x(xStart(i):xEnd(i)), order);
      end
      if order < 0
        yseg = y(xStart(i):xEnd(i));
      else
        [yseg,coeffs] = ltpda_polyreg(y(xStart(i):xEnd(i)), order);
      end
      [Sxxk,w] = ao.computeperiodogram(xseg,...
        win,options.nfft,esttype,options.Fs);
      [Syyk,w] =  ao.computeperiodogram(yseg,...
        win,options.nfft,esttype,options.Fs);
      [Sxyk,w] = ao.computeperiodogram({xseg,...
        yseg},win,options.nfft,esttype,options.Fs);
      Sxx  = Sxx + Sxxk;
      Syy  = Syy + Syyk;
```

```
        Sxy  = Sxy + Sxyk;
      end
  end

  Sxx = Sxx./k; % Average the sum of the periodograms
  if any(strcmpi(esttype,{'tfe','mscohere'})),
    Sxy = Sxy./k; % Average the sum of the periodograms

    if strcmpi(esttype,'mscohere'),
      Syy = Syy./k; % Average the sum of the periodograms
    end
  end

  % Generate the freq vector directly in Hz to avoid roundoff errors due to
  % conversions later.
  if ~freqVectorSpecified,
    w = psdfreqvec('npts',options.nfft, 'Fs',options.Fs);
  else
    if strcmpi(options.range,'onesided')
      warning(generatemsgid('InconsistentRangeOption'),...
        'Ignoring ''onesided'' option. When a frequency vector is specified, a ''twosided'' PSD is computed');
    end
    options.range = 'twosided';
  end


  % Compute the 1-sided or 2-sided PSD [Power/freq] or mean-square [Power].
  % Also, corresponding freq vector and freq units.
  [Pxx,w,units] = computepsd(Sxx,w,options.range,options.nfft,options.Fs,esttype);

  if any(strcmpi(esttype,{'tfe','mscohere'})),
    % Cross PSD.  The frequency vector and xunits are not used.
    [Pxy,f,xunits] = computepsd(Sxy,w,options.range,options.nfft,options.Fs,esttype);

    % Transfer function estimate.
    if strcmpi(esttype,'tfe'),
      Pxx = Pxy ./ Pxx; % Txy
    end

    % Magnitude Square Coherence estimate.
    if strcmpi(esttype,'mscohere'),
      % Auto PSD for 2nd input vector. The freq vector & xunits are not
      % used.
      [Pyy,f,xunits] = computepsd(Syy,w,options.range,options.nfft,options.Fs,esttype);
      Pxx = (abs(Pxy).^2)./(Pxx.*Pyy); % Cxy
    end
  end

  % Scale to required units
  [Pxx, info] = ao.welchscale(Pxx, win, fs, scale, inunits);
  info.navs = k;

  varargout = {Pxx, w, info};
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 segements.
0054   %if fix(k) ~= k),
0055   %   warning(generatemsgid('MustBeInteger'),'The number of segments is not an integer,
truncating data.');
0056   %end
0057
0058   k = fix(k);
0059 end
0060
0061 %-----------------------------------------------------------------------------------------
---------
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;
```

```
    end
    x = x{1};
    Lx = length(x);
  else
    if ~any(strcmpi(esttype,{'psd','ms'})),
      errid = generatemsgid('invalidSignalVectors');
      errmsg = 'You must specify a cell array with two signal vectors to estimate either the cross power spectral density or the transfer function.';
      return;
    end
  end

  x = x(:);
  isreal_x = isreal(x);

  % Parse 2nd input signal vector.
  if is2sig,
    y = y(:);
    isreal_x = isreal(y) && isreal_x;
    Ly = length(y);
    if Ly ~= Lx,
      errid = generatemsgid('invalidSignalVectors');
      errmsg = 'The length of the two input vectors must be equal to calculate the cross spectral density.';
      return;
    end
  end

  % Parse window and overlap, and cache remaining inputs.
  lenargin = length(varargin);
  if lenargin >= 1,
    win = varargin{1};
    if lenargin >= 2,
      noverlap = varargin{2};

      % Cache optional args nfft, fs, and spectrumType.
      if lenargin >= 3,  opts = varargin(3:end); end
    end
  end

  if isempty(win) | isscalar(win),
    winName = 'hamming';
    winParam = 'symmetric';
  end

end

%-----------------------------------------------------------------------------------------
%SEGMENT_INFO   Determine the information necessary to segment the input data.
%
%   Inputs:
%      M        - An integer containing the length of the data to be segmented
%      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)
%      NOVERLAP - An integer containing the number of samples to overlap (may be empty)
%
%   Outputs:
%      L        - An integer containing the length of the segments
%      NOVERLAP - An integer containing the number of samples to overlap
%      WIN      - A vector containing the window to be applied to each section
%      ERRID    - A string containing possible error identifier
%      ERRMSG   - A string containing possible error messages
%
%
%   The key to this function is the following equation:
%
%      K = (M-NOVERLAP)/(L-NOVERLAP)
%
%   where
%
%      K        - Number of segments
%      M        - Length of the input data X
%      NOVERLAP - Desired overlap
%      L        - Length of the segments
%
%   The segmentation of X is based on the fact that we always know M and two of the set
%   {K,NOVERLAP,L}, hence determining the unknown quantity is trivial from the above
%   formula.

function [L,noverlap,win,errid,errmsg] = segment_info(M,win,noverlap)

  % Initialize outputs
  L = [];
```

```
  errid = '';
  errmsg = '';

  % Check that noverlap is a scalar
  if any(size(noverlap) > 1),
    errid = generatemsgid('invalidNoverlap');
    errmsg = 'You must specify an integer number of samples to overlap.';
    return;
  end

  if isempty(win),
    % Use 8 sections, determine their length
    if isempty(noverlap),
      % Use 50% overlap
      L = fix(M./4.5);
      noverlap = fix(0.5.*L);
    else
      L = fix((M+7.*noverlap)./8);
    end
    % Use a default window
    win = hamming(L);
  else
    % Determine the window and its length (equal to the length of the segments)
    if ~any(size(win) <= 1) | ischar(win),
      errid = generatemsgid('invalidWindow');
      errmsg = 'The WINDOW argument must be a vector or a scalar.';
      return
    elseif length(win) > 1,
      % WIN is a vector
      L = length(win);
    elseif length(win) == 1,
      L = win;
      win = hamming(win);
    end
    if isempty(noverlap),
      % Use 50% overlap
      noverlap = fix(0.5.*L);
    end
  end

  % Do some argument validation
  if L > M,
    errid = generatemsgid('invalidSegmentLength');
    errmsg = 'The length of the segments cannot be greater than the length of the input signal.';
    return;
  end

  if noverlap >= L,
    errid = generatemsgid('invalidNoverlap');
    errmsg = 'The number of samples to overlap must be less than the length of the segments.';
    return;
  end
end

%-------------------------------------------------------------------------------
%WELCH_OPTIONS   Parse the optional inputs to the PWELCH function.
%   WELCH_OPTIONS returns a structure, OPTIONS, with 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

function [options,msg] = welch_options(isreal_x,N,varargin)

  % Generate defaults
  options.nfft = max(256,2^nextpow2(N));
  options.Fs = []; % Work in rad/sample

  % Determine if frequency vector specified
  freqVecSpec = false;
  if (length(varargin) > 0 && length(varargin{1}) > 1)
    freqVecSpec = true;
  end

  if isreal_x && ~freqVecSpec,
    options.range = 'onesided';
  else
    options.range = 'twosided';
  end
  msg = '';

  if any(strcmp(varargin, 'whole'))
    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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 $
%

function [yy, info] = welchscale(xx, win, fs, norm, inunits)

  nfft = length(win);
  S1   = sum(win);
  S2   = sum(win.^2);
  enbw = fs * S2 / (S1*S1);

  if isempty(norm)
    norm = 'None';
  end
  switch norm
    case 'ASD'
      yy = sqrt(xx);
      info.units = inunits ./ unit('Hz^0.5');
    case 'PSD'
      yy = xx;
      info.units = inunits.^2/unit('Hz');
    case 'AS'
      yy = sqrt(xx * enbw) ;
      info.units = inunits;
    case 'PS'
      yy = xx * enbw;
      info.units = inunits.^2;
    case 'None'
      yy = xx;
      info.units = inunits;
    otherwise
      error('Unknown normalisation');
```

```
  end

  info.nfft = nfft;
  info.enbw = enbw;
  info.norm = norm;

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

X

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 ^

```
% X Get the data property 'x'.
%%%
```

```
%
% HISTORY:      11-07-07 M Hewitson
%                 Creation.
%
%%%
```

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 ^

```
% XSPEC performs cross-spectral analysis of various forms.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
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 %%%%
```

```
%---------------- check input parameters

% points in FFT
Nfft  = find(pl, 'Nfft');
if ~isempty(Nfft)
  if Nfft < 0
    Nfft = abs(Nfft) * lmin;
    utils.helper.msg(msg.PROC2, 'using default Nfft of %g', Nfft);
    pl.pset('Nfft', Nfft);
  end
end

% Window object
Win   = find(pl, 'Win');
if ~isempty(Win)
  if length(Win.win)~=Nfft
    switch Win.type
      case 'Kaiser'
        Win = specwin(Win.type, Nfft, Win.psll);
      otherwise
        Win = specwin(Win.type, Nfft);
    end
    utils.helper.msg(msg.PROC2, 'reset window to %s(%d)', strrep(Win.type, '_', '\_'), length(Win.win));
    pl.pset('Win', Win);
  end
end

% desired overlap
Olap = find(pl, 'Olap');
if ~isempty(Olap)
  if Olap < 0
    % use window function rov
    Olap = Win.rov;
    utils.helper.msg(msg.PROC2, 'using recommended overlap of %2.1f%%', Olap);
    pl.pset('Olap', Olap);
  end
end

% desired detrending order
order = find(pl, 'Order');
if isempty(order)
  order = 0;
  utils.helper.msg(msg.PROC1, 'using default detrending order 0 (mean)');
  pl.append('Order', order);
end

% Loop over input AOs
bs(na,na) = ao;
for j=1:na
  for k=1:na
    if k>=j
      utils.helper.msg(msg.PROC1, 'computing (%d,%d) %s -> %s', j, k, invars{j}, invars{k});

      % -------- Make Xspec estimate

      % Compute xspec using welch
      [txy, f, info] = ao.welch(as(j), as(k), method, pl);

      % Keep the data shape of the first input AO
      if size(as(1).data.y,1) == 1
        txy = txy.';
        f   = f.';
      end

      % create new output fsdata
      fs = fsdata(f, txy, fsmax);
      fs.setEnbw(info.enbw);
      fs.setXunits('Hz');
      fs.setYunits(info.units);
      fs.setNavs(info.navs);

      % make output analysis object
      bs(j,k) = ao(fs);

      %----------- Add history

      % we need to get the input histories in the same order as the inputs
      % to this function call, not in the order of the input to tfestimate;
      % otherwise the resulting matrix on a 'create from history' will be
      % mirrored.
      if j<k
        bs(j,k).addHistory(mi, pl, [invars(j) invars(k)], [as(j).hist, as(k).hist]);
      else
```

```
            bs(j,k).addHistory(mi, pl, [invars(j) invars(k)], [as(k).hist, as(j).hist]);
          end

          % set name
          bs(j,k).setName(sprintf('%s(%s->%s)', method, invars{j}, invars{k}),
'internal');

        end % End upper triangle
      end % End second loop over AOs
    end % End first loop over AOs

    % Fill remainder
    for j=1:na
      for k=1:na
        if k<j
          % Do we need a copy here? Perhaps
          % copying the pointer is enough...
          bs(j,k) = bs(k,j);
        end
      end
    end

    % Set output
    varargout{1} = bs;
  end

```

*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 ^

```
% 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
%
%%%
```

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.
- lxspec 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 ^

```
% Y Get the data property 'y'.
%%%
```

```
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, 'ao', '', utils.const.categories.internal, '$Id: y.m,v 1.1 2008/09/05 13:55:17 ingo Exp $', sets, pl);
end

%%%
```

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 ^

```
% 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
%
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function objs = fromFile(obj, pli)

  VERSION = '$Id: fromFile.m,v 1.2 2008/09/08 09:40:46 hewitson Exp $';

  % Which file type are we dealing with?
  if ischar(pli)
    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
0055      case '.xml'
0056        root_node = xmlread(filename);
0057        objs = utils.helper.xmlread(root_node, class(obj));
0058
0059      otherwise
0060        error('### Unknown file type [%s].', ext(2:end));
0061    end
0062
0063 end
0064
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromRepository

PURPOSE ^

Construct a plist filter from a repository

SYNOPSIS ^

function fs = fromRepository(f, pli)

DESCRIPTION ^

```
 Construct a plist filter from a repository
%%%
```

```
0024
0025   % Add default values
0026   pl = combine(pli, ii.plists);
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
0073   end
0074
0075   % close connection
0076   if closeConn
0077     close(conn);
0078   end
0079 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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct = rmfield(obj_struct, 'plist');
    obj_struct.creator = provenance('updater 1.0');
  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (plist) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');
  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (param) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

abcascade

PURPOSE ^

ABCASCADE Cascade two filters together to get a new filter.

SYNOPSIS ^

function [ao,bo] = abcascade(a1,b1,a2,b2)

DESCRIPTION ^

```
 ABCASCADE Cascade two filters together to get a new filter.
%%%
```

```
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 pzm = fromLISO(pzm, pli)

  import utils.const.*

  VERSION = '$Id: fromLISO.m,v 1.3 2008/08/22 14:17:27 ingo Exp $';

  ii = pzm.getInfo(class(pzm), 'From LISO File');

  % Set the method version string in the minfo object
  ii.setMversion([VERSION '-->' ii.mversion]);

  filename = find(pli, 'filename');

  poles = [];
  zeros = [];
  gain  = [];
  % Open the file for reading
  fd = fopen(filename, 'r');
  utils.helper.msg(msg.OPROC2, ['reading ' filename]);
  while ~feof(fd)
    line = fgetl(fd);
    % get first token
    [s,r] = strtok(line);
    if strcmp(s, 'pole')
      % get next token as frequency
      [s,r] = strtok(r);
      pf    = lisoStr2Num(s);
      % does this pole have a Q?
      pq = NaN;
      if ~isempty(r)
        [s,r] = strtok(r);
        if isnumeric(lisoStr2Num(s))
          pq = lisoStr2Num(s);
        end
      end
      % make pole
      if isnan(pq)
        p = pz(pf);
        utils.helper.msg(msg.OPROC2, 'found pole: %g', pf);
      else
        p = pz(pf, pq);
        utils.helper.msg(msg.OPROC2, 'found pole: %g, %g', pf, pq);
      end
      poles = [poles p];
    elseif strcmp(s, 'zero')
      % get next token as frequency
      [s,r] = strtok(r);
      zf    = lisoStr2Num(s);
      % does this pole have a Q?
      zq = NaN;
      if ~isempty(r)
        [s,r] = strtok(r);
        if isnumeric(lisoStr2Num(s))
          zq = lisoStr2Num(s);
        end
      end
      % make zero
      if isnan(zq)
        z = pz(zf);
        utils.helper.msg(msg.OPROC2, 'found zero: %g', zf);
      else
        z = pz(zf, zq);
        utils.helper.msg(msg.OPROC2, 'found zero: %g, %g', zf, zq);
      end
      zeros = [zeros z];
    elseif strcmp(s, 'factor')
      % get next token as gain
      [s, r] = strtok(r);
      gain   = lisoStr2Num(s);
      utils.helper.msg(msg.OPROC2, 'found factor: %g', gain);
    end
  end
  % Close file
  fclose(fd);
  % get model name
  [path, name, ext, vers] = fileparts(filename);
  % set object
  pzm.gain  = gain;
  pzm.poles = poles;
  pzm.zeros = zeros;
  pzm.name  = name;

  pzm.addHistory(ii, pli, [], []);

end

```

```
%%%%%%%%%%% A function to convert LISO number strings to doubles %%%%%%%%%%

function d = lisoStr2Num(s)

  s = strrep(s, 'm', 'e-3');
  s = strrep(s, 'u', 'e-6');
  s = strrep(s, 'n', 'e-9');
  s = strrep(s, 'p', 'e-12');
  s = strrep(s, 'f', 'e-15');
  s = strrep(s, 'k', 'e3');
  s = strrep(s, 'M', 'e6');
  s = strrep(s, 'G', 'e9');

  d = str2double(s);

end % function d = lisoStr2Num(s)
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromPolesAndZeros

PURPOSE ^

**%%%

```
%               pl   - plist
%
% HISTORY:      07-05-2007 Hewitson
%               Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function pzm = fromPolesAndZeros(pzm, pli)

  VERSION = '$Id: fromPolesAndZeros.m,v 1.9 2008/09/04 13:37:14 ingo Exp $';


  % get pzmodel info
  ii = pzmodel.getInfo('pzmodel', 'From Poles/Zeros');

  % Set the method version string in the minfo object
  ii.setMversion([VERSION '-->' ii.mversion]);

  % Combine input plist with default values
  pl = combine(pli, ii.plists);

  % Set fields
  pzm.gain  = find(pl, 'gain');
  ps = find(pl, 'poles');
  zs = find(pl, 'zeros');

  % If the poles/zeros in the plist are no pz-objects then convert them
  if ~isa(ps, 'pz') && ~isempty(ps)
    ps = pz(ps);
  end
  if ~isa(zs, 'pz') && ~isempty(zs)
    zs = pz(zs);
  end

  % Add only valid poles
  pzm.poles = [];
  for kk = 1:length(ps)
    if ~isnan(ps(kk).f)
      pzm.poles = [pzm.poles ps(kk)];
    end
  end

  % Add only valid zeros
  pzm.zeros = [];
  for kk = 1:length(zs)
    if ~isnan(zs(kk).f)
      pzm.zeros = [pzm.zeros zs(kk)];
    end
  end

  name = find(pl, 'name');
  if ~isempty(name)
    pzm.name = name;
  end

  % Add history
  pzm.addHistory(ii, pl, [], []);

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function obj = setGain(obj, val)
  %%% decide whether we modify the pzmodel-object, or create a new one.
  obj = copy(obj, nargout);

  %%% set 'gain'
  obj.gain = val;
end

```

*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'
%%%
```

*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'
%%%
```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

    if isempty(obj_struct.plist)
      obj_struct.plist = plist();
    end
    ii = minfo('update_struct', 'pzmodel', '', '', ...
               '$Id: update_struct.m,v 1.1 2008/07/10 12:06:50 ingo Exp $', ...
               {'Default'}, plist(), 0, 0);
    obj_struct.hist = history(time().utc_epoch_milli, ii, obj_struct.plist);
    obj_struct = rmfield(obj_struct, 'plist');

    obj_struct.creator = provenance('Updater 1.0 -> 1.1');

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (pzmodel) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

< Master index Index for m/sigproc/time_domain >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

< Master index Index for m/sigproc/frequency_domain >

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_ltfe	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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

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.
%%%
```

```
    w    = varargin{1};
    name = varargin{2};
    N    = varargin{3};
    PSLL = varargin{4};
  else
    error('### Incorrect inputs.');
  end

  % Get window
  if isempty(PSLL)
    % get standard window
    z = [0:N-1]./N;
    win_name = sprintf('win_%s', name);
    try
      w = feval(win_name, w, z);
    catch
      error('\n### Your window [%s] is not a supported window.\n### Please type >> specwin.getTypes to see all supported windows.', win_name(5:end));
    end

  else

    % Deal with Kaiser
    w = get_kaiser_window(w, N, PSLL);

  end
end

%-------------------------------------------------------------------------
% Make Kaiser window
%
function w = get_kaiser_window(w, N, psll)

  % Make window struct
  w.type     = 'Kaiser';
  w.alpha    = specwin.kaiser_alpha(psll);
  w.psll     = psll;
  w.rov      = specwin.kaiser_rov(w.alpha);
  w.nenbw    = specwin.kaiser_nenbw(w.alpha);
  w.w3db     = specwin.kaiser_w3db(w.alpha);
  w.flatness = specwin.kaiser_flatness(w.alpha);

  % Compute window samples
  % - here we adjust to make the window asymmetric
  v = kaiser(N+1, pi*w.alpha);
  v = v(1:end-1);
  w.ws       = sum(v);
  w.ws2      = sum(v.*v);
  w.win      = v.';
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

kaiser_alpha

PURPOSE ^

KAISER_ALPHA returns the alpha parameter that gives the required input

SYNOPSIS ^

function alpha = kaiser_alpha(req_psll)

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 ^

```
function alpha = kaiser_alpha(req_psll)

% 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 $
%


a0 = -0.0821377;
a1 = 4.71469;
a2 = -0.493285;
a3 = 0.0889732;

x = req_psll / 100;
alpha = (((((a3 * x) + a2) * x) + a1) * x + a0);


% END
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function flatness = kaiser_flatness(alpha)

% 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 $
%

a0       = 0.141273;
a1       = 0.262425;
a2       = 0.00642551;
a3       = -0.000405621;
x        = alpha;
flatness =  -1. / (((((a3 * x) + a2) * x) + a1) * x + a0);

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function nenbw = kaiser_nenbw(alpha)

% 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 $
%

a0    = 0.768049;
a1    = 0.411986;
a2    = -0.0264817;
a3    = 0.000962211;
x     = alpha;
nenbw =  (((((a3 * x) + a2) * x) + a1) * x + a0);

% 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 ^

```
function rov = kaiser_rov(alpha)

% 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 $
%


a0  = 0.0061076;
a1  = 0.00912223;
a2  = -0.000925946;
a3  = 4.42204e-05;
x   = alpha;
rov =  100 - 1 / (((((a3 * x) + a2) * x) + a1) * x + a0);

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w3db = kaiser_w3db(alpha)

% 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 $
%

a0   = 0.757185;
a1   = 0.377847;
a2   = -0.0238342;
a3   = 0.00086012;
x    = alpha;
w3db = (((((a3 * x) + a2) * x) + a1) * x + a0);

% 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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct.type = obj_struct.name;
    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');
    obj_struct = rmfield(obj_struct, 'plist');
  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (specwin) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_BH92(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.35875 - 0.48829 * cos (z) + 0.14128 * cos (2 * z) - 0.01168 * cos (3 * z);

% Make window struct
w.type     = 'BH92';
w.alpha    = 0;
w.psll     = 92;
w.rov      = 66.1;
w.nenbw    = 2.0044;
w.w3db     = 1.8962;
w.flatness = -0.8256;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% 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 ^

```
function w = win_Bartlett(w, z)

% 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 $
%


v = z * 2;
idx = find(v>1);
v(idx) = 2-v(idx);


% Make window struct
w.type     = 'Bartlett';
w.alpha    = 0;
w.psll     = 26.5;
w.rov      = 50;
w.nenbw    = 1.3333;
w.w3db     = 1.2736;
w.flatness = -1.8242;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;



% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_FTHP(w, z)

% 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 $
%

z = pi * (2 * z - 1);
v =  (1 + 1.912510941 ...
        * cos (z) + 1.079173272 * cos (2 * z) + ...
        0.1832630879 * cos (3 * z));

% Make window struct
w.type     = 'FTHP';
w.alpha    = 0;
w.psll     = 70.4;
w.rov      = 72.3;
w.nenbw    = 3.4279;
w.w3db     = 3.3846;
w.flatness = 0.0096;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_FTNI(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.2810639 - 0.5208972 * cos (z) + 0.1980399 * cos (2 * z);

% Make window struct
w.type     = 'FTNI';
w.alpha    = 0;
w.psll     = 44.4;
w.rov      = 65.6;
w.nenbw    = 2.9656;
w.w3db     = 2.9355;
w.flatness = 0.0169;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% 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 ^

```
function w = win_FTSRS(w, z)

% 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 $
%

z = z * 2 * pi;
v =  (1.0 - 1.93 * cos (z) + 1.29 ...
      * cos (2 * z) - 0.388 * cos (3 * z) + 0.028 * cos (4 * z));

% Make window struct
w.type     = 'FTSRS';
w.alpha    = 0;
w.psll     = 76.6;
w.rov      = 75.4;
w.nenbw    = 3.7702;
w.w3db     = 3.7274;
w.flatness = -0.0156;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_HFT116D(w, z)

% 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 $
%

z = z * 2 * pi;
v = (1 - 1.9575375 * cos (z) + ...
      1.4780705 * cos (2 * z) - 0.6367431 * cos (3 * z) + ...
      0.1228389 * cos (4 * z) - 0.0066288 * cos (5 * z));

% Make window struct
w.type     = 'HFT116D';
w.alpha    = 0;
w.psll     = 116.8;
w.rov      = 78.2;
w.nenbw    = 4.2186;
w.w3db     = 4.1579;
w.flatness = -0.0028;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_HFT144D(w, z)

% 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 $
%

z = z * 2 * pi;
v = (1 - 1.96760033 * cos (z) ...
      + 1.57983607 * cos (2 * z) - 0.81123644 * cos (3 * z) + ...
      0.22583558 * cos (4 * z) - 0.02773848 * cos (5 * z) + ...
      0.00090360 * cos (6 * z));

% Make window struct
w.type     = 'HFT144D';
w.alpha    = 0;
w.psll     = 144.1;
w.rov      = 79.9;
w.nenbw    = 4.5386;
w.w3db     = 4.4697;
w.flatness = 0.0021;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_HFT169D(w, z)

% 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 $
%

z = z * 2 * pi;
v = (1 - 1.97441843 * cos (z) ...
      + 1.65409889 * cos (2 * z) - 0.95788187 * cos (3 * z) + ...
      0.33673420 * cos (4 * z) - 0.06364622 * cos (5 * z) + ...
      0.00521942 * cos (6 * z) - 0.00010599 * cos (7 * z));

% Make window struct
w.type     = 'HFT169D';
w.alpha    = 0;
w.psll     = 169.5;
w.rov      = 81.2;
w.nenbw    = 4.8347;
w.w3db     = 4.7588;
w.flatness = 0.0017;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_HFT196D(w, z)

% 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 $
%

z = z * 2 * pi;
v = (1 - 1.979280420 * cos (z) ...
      + 1.710288951 * cos (2 * z) - 1.081629853 * cos (3 * z) + ...
      0.448734314 * cos (4 * z) - 0.112376628 * cos (5 * z) + ...
      0.015122992 * cos (6 * z) - 0.000871252 * cos (7 * z) + ...
      0.000011896 * cos (8 * z));

% Make window struct
w.type     = 'HFT196D';
w.alpha    = 0;
w.psll     = 196.2;
w.rov      = 82.3;
w.nenbw    = 5.1134;
w.w3db     = 5.0308;
w.flatness = 0.0013;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_HFT223D(w, z)

% 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 $
%

z = z * 2 * pi;
v = (1 - 1.98298997309 * cos(z) + ...
    1.75556083063 * cos (2 * z) - 1.19037717712 * cos (3 * z) + ...
      0.56155440797 * cos (4 * z) - 0.17296769663 * cos (5 * z) + ...
      0.03233247087 * cos (6 * z) - 0.00324954578 * cos (7 * z) + ...
      0.00013801040 * cos (8 * z) - 0.00000132725 * cos (9 * z));

% Make window struct
w.type     = 'HFT223D';
w.alpha    = 0;
w.psll     = 223;
w.rov      = 83.3;
w.nenbw    = 5.3888;
w.w3db     = 5.3;
w.flatness = -0.0011;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% 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 ^

```
function w = win_HFT248D(w, z)

% 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 $
%

z = z * 2 * pi;
v = (1 - 1.985844164102 * cos(z) + ...
    1.791176438506 * cos (2 * z) - 1.282075284005 * cos (3 * z) + ...
      0.667777530266 * cos (4 * z) - 0.240160796576 * cos (5 * z) + ...
      0.056656381764 * cos (6 * z) - 0.008134974479 * cos (7 * z) + ...
      0.000624544650 * cos (8 * z) - 0.000019808998 * cos (9 * z) + ...
      0.000000132974 * cos (10 * z));

% Make window struct
w.type     = 'HFT248D';
w.alpha    = 0;
w.psll     = 248.4;
w.rov      = 84.1;
w.nenbw    = 5.6512;
w.w3db     = 5.5567;
w.flatness = 0.0009;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_HFT70(w, z)

% 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 $
%

z = z * 2 * pi;
v =  (1 - 1.90796 * cos (z) + ...
      1.07349 * cos (2 * z) - 0.18199 * cos (3 * z));

% Make window struct
w.type     = 'HFT70';
w.alpha    = 0;
w.psll     = 70.4;
w.rov      = 72.2;
w.nenbw    = 3.4129;
w.w3db     = 3.3720;
w.flatness = -0.0065;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_HFT90D(w, z)

% 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 $
%

z = z * 2 * pi;
v =  (1 - 1.942604 * cos (z) + ...
      1.340318 * cos (2 * z) - 0.440811 * cos (3 * z) + ...
      0.043097 * cos (4 * z));

% Make window struct
w.type     = 'HFT90D';
w.alpha    = 0;
w.psll     = 90.2;
w.rov      = 76;
w.nenbw    = 3.8832;
w.w3db     = 3.8320;
w.flatness = -0.0039;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% 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 ^

```
function w = win_HFT95(w, z)

% 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 $
%

z = z * 2 * pi;
v = (1 - 1.9383379 * cos (z) + ...
      1.3045202 * cos (2 * z) - 0.4028270 * cos (3 * z) + ...
      0.0350665 * cos (4 * z));

% Make window struct
w.type     = 'HFT95';
w.alpha    = 0;
w.psll     = 95;
w.rov      = 75.6;
w.nenbw    = 3.8112;
w.w3db     = 3.759;
w.flatness = 0.0044;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Hamming(w, z)

% 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 $
%

v = 0.54 - 0.46 * cos(2*pi*z);

% Make window struct
w.type     = 'Hamming';
w.alpha    = 0;
w.psll     = 42.7;
w.rov      = 50;
w.nenbw    = 1.3628;
w.w3db     = 1.3008;
w.flatness = -1.7514;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Hanning(w, z)

% 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 $
%

v = 0.5 * (1 - cos(2*pi*z));

% Make window struct
w.type     = 'Hanning';
w.alpha    = 0;
w.psll     = 31.5;
w.rov      = 50;
w.nenbw    = 1.5;
w.w3db     = 1.4382;
w.flatness = -1.4236;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% 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 ^

```
function w = win_Nuttall3(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.375 - 0.5 * cos (z) + 0.125 * cos (2 * z);

% Make window struct
w.type     = 'Nuttall3';
w.alpha    = 0;
w.psll     = 46.7;
w.rov      = 64.7;
w.nenbw    = 1.9444;
w.w3db     = 1.8496;
w.flatness = -0.8630;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Nuttall3a(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.40897 - 0.5 * cos (z) + 0.09103 * cos (2 * z);

% Make window struct
w.type     = 'Nuttall3a';
w.alpha    = 0;
w.psll     = 64.2;
w.rov      = 61.2;
w.nenbw    = 1.7721;
w.w3db     = 1.6828;
w.flatness = -1.0453;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Nuttall3b(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.4243801 - 0.4973406 * cos (z) + 0.0782793 * cos (2 * z);

% Make window struct
w.type     = 'Nuttall3b';
w.alpha    = 0;
w.psll     = 71.5;
w.rov      = 59.8;
w.nenbw    = 1.7037;
w.w3db     = 1.6162;
w.flatness = -1.1352;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Nuttall4(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.3125 - 0.46875 * cos(z) + 0.1875 * cos (2 * z) - 0.03125 * cos (3 * z);

% Make window struct
w.type     = 'Nuttall4';
w.alpha    = 0;
w.psll     = 60.9;
w.rov      = 70.5;
w.nenbw    = 2.31;
w.w3db     = 2.1884;
w.flatness = -0.6184;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% 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 ^

```
function w = win_Nuttall4a(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.338946 - 0.481973 * cos (z) + 0.161054 * cos (2 * z) - 0.018027 * cos (3 * z);

% Make window struct
w.type     = 'Nuttall4a';
w.alpha    = 0;
w.psll     = 82.6;
w.rov      = 68;
w.nenbw    = 2.1253;
w.w3db     = 2.0123;
w.flatness = -0.7321;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Nuttall4b(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.355768 - 0.487396 * cos (z) + 0.144232 * cos (2 * z) - 0.012604 * cos (3 * z);

% Make window struct
w.type     = 'Nuttall4b';
w.alpha    = 0;
w.psll     = 93.3;
w.rov      = 66.3;
w.nenbw    = 2.0212;
w.w3db     = 1.9122;
w.flatness = -0.8118;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Nuttall4c(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.3635819 - 0.4891775 * cos (z) + 0.1365995 * cos (2 * z) - 0.0106411 * cos (3 * z);

% Make window struct
w.type     = 'Nuttall4c';
w.alpha    = 0;
w.psll     = 98.1;
w.rov      = 65.6;
w.nenbw    = 1.9761;
w.w3db     = 1.8687;
w.flatness = -0.8506;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Rectangular(w, z)

% 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 $
%

v = ones(size(z));

% Make window struct
w.type     = 'Rectangular';
w.alpha    = 0;
w.psll     = 13.3;
w.rov      = 0.0;
w.nenbw    = 1.0;
w.w3db     = 0.8845;
w.flatness = -3.9224;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% 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 ^

```
function w = win_SFT3F(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.26526 - 0.5 * cos (z) + 0.23474 * cos (2 * z);

% Make window struct
w.type     = 'SFT3F';
w.alpha    = 0;
w.psll     = 31.7;
w.rov      = 66.7;
w.nenbw    = 3.1681;
w.w3db     = 3.1502;
w.flatness = 0.0082;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_SFT3M(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.28235 - 0.52105 * cos (z) + 0.19659 * cos (2 * z);

% Make window struct
w.type     = 'SFT3M';
w.alpha    = 0;
w.psll     = 44.2;
w.rov      = 65.5;
w.nenbw    = 2.9452;
w.w3db     = 2.9183;
w.flatness = -0.0115;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_SFT4F(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.21706 - 0.42103 * cos (z) + 0.28294 * cos (2 * z) - 0.07897 * cos (3 * z);

% Make window struct
w.type     = 'SFT4F';
w.alpha    = 0;
w.psll     = 44.7;
w.rov      = 75;
w.nenbw    = 3.7970;
w.w3db     = 3.7618;
w.flatness = 0.0041;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_SFT4M(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.241906 - 0.460841 * cos(z) ...
  + 0.255381 * cos (2 * z) - 0.041872 * cos (3 * z);

% Make window struct
w.type     = 'SFT4M';
w.alpha    = 0;
w.psll     = 66.5;
w.rov      = 72.1;
w.nenbw    = 3.3868;
w.w3db     = 3.3451;
w.flatness = -0.0067;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_SFT5F(w, z)

% 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 $
%

z = z * 2 * pi;
v = 0.1881 - 0.36923 * cos (z) + ...
    0.28702 * cos (2 * z) - 0.13077 * cos (3 * z) + 0.02488 * cos (4 * z);

% Make window struct
w.type     = 'SFT5F';
w.alpha    = 0;
w.psll     = 57.3;
w.rov      = 78.5;
w.nenbw    = 4.3412;
w.w3db     = 4.2910;
w.flatness = -0.0025;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_SFT5M(w, z)

% 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 $
%

z = z * 2 * pi;
v =  0.209671 - 0.407331 * cos(z) + ...
    0.281225 * cos (2 * z) - 0.092669 * cos (3 * z) + ...
    0.0091036 * cos (4 * z);

% Make window struct
w.type     = 'SFT5M';
w.alpha    = 0;
w.psll     = 89.9;
w.rov      = 76;
w.nenbw    = 3.8852;
w.w3db     = 3.8340;
w.flatness = 0.0039;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function w = win_Welch(w, z)

% 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 $
%

v = 1 - (2 * z - 1).^2;

% Make window struct
w.type     = 'Welch';
w.alpha    = 0;
w.psll     = 21.3;
w.rov      = 29.3;
w.nenbw    = 1.2;
w.w3db     = 1.1535;
w.flatness = -2.2248;
w.ws       = sum(v);
w.ws2      = sum(v.*v);
w.win      = v;

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

- 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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef ltpda_obj < handle

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                            %
```

```
0056    %%%%
```

```
  methods (Abstract)
    varargout = char(varargin)
    varargout = copy(varargin)
    txt       = display(varargin)
  end

  methods (Abstract = true, Static = true)
    varargout = update_struct(varargin);
  end

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% LTPDA_UOH is the abstract ltpda base class for ltpda user object classes with history
%%%
```

```
%---------- Public (read/write) Properties  ----------
properties
end

%---------- Protected read-only Properties ----------
properties (SetAccess = protected)
  hist    = '';
end

%---------- Private Properties ----------
properties (GetAccess = protected, SetAccess = protected)
end

%---------- Abstract Properties ----------
properties (Abstract = true, SetAccess = protected)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              Check property setting                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

methods
  function obj = set.hist(obj, val)
    if ~(isa(val, 'history') || isempty(val))
      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));
    end
    obj.hist = val;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Constructor                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

methods
  function obj = ltpda_uoh(varargin)

    %%% Call superclass
    obj = obj@ltpda_uo(varargin{:});

    %%%%%%%%%%   Set dafault values   %%%%%%%%%%

    if nargin == 1

      if isstruct(varargin{1})
        %%%%%%%%%%   obj = ltpda_uoh(struct)   %%%%%%%%%%
        %%% Set properties which are declared in this class
        obj.hist = utils.helper.struct2obj(varargin{1}.hist, 'history');
      end

    end

  end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 Methods (public)                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

methods
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 Methods (protected)                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

methods (Access = protected)
  varargout = setHist(obj, val)
  obj = fromFile(pbj, pli)
  obj = fromRepository(obj, pli)
  obj = fromLISO(obj, filename)
  obj = fromDataInMAT(obj, data, filename)
  obj = fromDatafile(obj, pli)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 Methods (public)                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

methods
  varargout = addHistory(varargin)
  varargout = setName(varargin)
end
```

```
0139
0140    %%%%
```

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 ^

```
% 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
%
```

```
%%%
```

```
  %%%%
```

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 ^

```
% 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
%%%
```

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 ^

```
% 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
%%%
```

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
%%%
```

```
%       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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function c = cell_mult(a,b)
n1 = size(a,1);
n2 = size(a,2);
n3 = size(b,2);
c = cell(n1,n3);
for i=1:n1
  for j=1:n3
    for k=1:n2
      if ~(isequal(a{i,k}, []) || isequal(b{k,j}, []))

        if isempty(c{i,j})
          c{i,j} = a{i,k}*b{k,j};
        else
          try
            c{i,j} = c{i,j} + a{i,k}*b{k,j};
          catch
            c{i,j} = sym(c{i,j}) + sym(a{i,k})*sym(b{k,j});
          end
        end
      end
    end
  end
end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
%       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
%%%
```

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 blck 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 blck 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.
```

```
%                blocs may be empty
%
% ***** There are no parameters *****
%
% VERSION: '$Id: $'
%
%
% HISTORY:
% 12-08-2008
%
% TO DO :
%%%
```

```
%% assigning content
cell_mat_out = cell(Nrows, Ncols);
for i=1:Nrows
  for j=1:Ncols
    if ~isequal(cell_mat_in{i,j}, [])
      if ~isempty(rows_log{i}) && ~isempty(cols_log{j})
        cell_mat_out{i,j} = cell_mat_in{i,j}(rows_log{i},cols_log{j});
      else
        switch class(cell_mat_in{i,j})
          case {'double' 'sym'}
            cell_mat_out{i,j} = zeros( sum(rows_log{i}>0), sum(cols_log{j}>0));
          case 'cell'
            cell_mat_out{i,j} = cell( sum(rows_log{i}>0), sum(cols_log{j}>0));
        end
      end
    end
  end
end

rowsizes_out = zeros(1, Nrows);
for i=1:Nrows
  if isa(rows_log{i},'double')
    rowsizes_out(i) = length(rows_log{i});
  else
    rowsizes_out(i) = sum(rows_log{i});
  end
end

colsizes_out = zeros(1, Ncols);
for i=1:Ncols
  if isa(cols_log{i},'double')
    colsizes_out(i) = length(cols_log{i});
  else
    colsizes_out(i) = sum(cols_log{i});
  end
end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
%                 '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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = cellstrfind(c,s,option)
  value = strcmp(c,s);
  position = find(value);
  sum = length(position);
  if strcmp(option, 'sum')
    varargout = {sum};
  elseif strcmp(option, 'value')
    varargout = {value};
  elseif strcmp(option, 'position')
    varargout = {position};
  elseif strcmp(option, 'all')
    varargout = {position, sum, value};
  end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ssmFromBuiltinSystem

PURPOSE ^

**%%%

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = ssmFromBuiltinSystem(pli)

  VERSION = '$Id: ssmFromBuiltinSystem.m,v 1.41 2008/09/04 13:37:15 ingo Exp $';

  utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

  % get AO info
  ii = ssm.getInfo('ssm', 'From Built-in Model');

  % Set the method version string in the minfo object
  ii.setMversion([VERSION '-->' ii.mversion]);

  % Get default params
  pl = combine(pli, ii.plists);

  % Get the name of the model the user wants
  bsys = find(pl, 'Built-in');

  % Get a list of user model directories
  paths = getappdata(0, 'ltpda_ssm_model_paths');
  if ischar(paths)
    paths = {paths};
  end

  % Always look in the system directory
  loc = mfilename('fullpath');
  idx = strfind(loc, filesep);
  loc = loc(1:idx(end));
  loc = fullfile(loc, '../../m/ssm_models');
  paths = [paths {loc}];

  for jj=1:numel(paths)
    utils.helper.msg(utils.const.msg.PROC1, 'looking for models in %s', paths{jj});
  end

  % list files in here
  models = {};
  for pp = 1:numel(paths)
    files = utils.prog.filescan(paths{pp}, '.m');
    for ff = 1:numel(files)
      [path, mname, ext, vers] = fileparts(files{ff});
      if strncmp(mname, 'ssm_model_', 10)
        models = [models {strrep(mname, 'ssm_model_', '')}];
      end
    end
  end

  % Give a list and an error if bsys is empty
  if isempty(bsys)
    for k=1:numel(models)
      disp(sprintf('%03d: %s', k, models{k}));
    end
    error('### No model specified. Please choose one from the list above.');
```

```
  end

  % Find the matching model
  sys = [];
  if any(strcmp(models, bsys)) || (isnumeric(bsys) && bsys <= numel(models))
    if isnumeric(bsys)
      fcnname = ['ssm_model_' models{bsys}];
    else
      fcnname = ['ssm_model_' bsys];
    end
    [sys pl_dim_red] = feval(fcnname, pl);
    sys.version = ii.mversion;
    sys.created = time(0);
    sys.creator = provenance();
    sys.hist    = history(time().utc_epoch_milli, ii, pl, [], []);

    % Convert to SSM object
    sys = ssm(sys);
    sys.reduce_model(pl_dim_red)

  else
    for k=1:numel(models)
      disp(sprintf('%03d: %s', k, models{k}));
    end
    error('### Model not found. Please choose one from the above list.');
  end

  % Set output
  varargout{1} = sys;

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ssmFromDescription

PURPOSE ^

**%%%

```
%
% 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 $';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sys = ssmFromDescription(pli)
utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

% Get default params
pl = combine(pli, ssm.getDefaultPlist('From Description'));
if ~isa(pl,'plist') % checking input type
   error(['error because input is not a plist but a ', class(pl)]);
end

sys(length(pl) ) = ssm;
for i_pl = 1:length(pl) % going through the input
   % filling compulsory user defined fields
   userfields = {'name','ssnames', 'outputnames', 'inputnames', 'timestep', 'amats', ...
     'bmats', 'cmats', 'dmats'};
   % other optional fields that may be used defined
   otherfields =  {'mmats' 'amats_handles' 'bmats_handles' 'cmats_handles' 'dmats_handles'...
     'inputvarnames'  'ssvarnames' 'outputvarnames' ...
     'paramnames' 'paramvalues' 'paramsigmas'};

   for f = userfields
      if isparam(pl(i_pl), f{1})
         sys(i_pl).(f{1}) = find(pl(i_pl),f{1});
      else
         display(['###  ERROR : field in ssm named ''',f{1},''' must be user defined  ###']);
         display('###        list of other compulsory user defined  fields        ###')
         display(char(userfields))
         display('###         list of other optional user defined  fields         ###')
         display(char(otherfields))
         error(['see above message and lists ^^ ']);
      end
   end

   for f = otherfields
      if isparam(pl(i_pl),f{1})
         sys(i_pl).(f{1}) = find(pl(i_pl),f{1});
      end
   end
   % fields that do not depend on input variables.
   sys(i_pl).addHistory(ssm.getInfo('ssm', 'From Description'), pli(i_pl), {''}, []);

end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ssmFromFilename

PURPOSE ^

**%%%

```
%
% HISTORY:     15-04-2008 Hewitson
%              Creation
% VERSION :    '$Id: ssmFromFilename.m,v 1.11 2008/08/08 15:32:33 adrien Exp $';
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sys = ssmFromFilename(pli)
utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

% Check the filename and extension
filename = find(pli, 'filename');
[pathstr, f_name, ext] = fileparts(filename);

% Add default values
switch ext
  case '.mat'
    in = load(filename);
    sys = in.objs;
    sys.addHistory(ssm.getInfo('ssm', 'From MAT File'), pli, {''}, []);
  case '.xml'
    root_node = xmlread(filename);
    sys = ltpda_xmlread(root_node, 'ssm');
    sys.addHistory(ssm.getInfo('ssm', 'From XML File'), pli, {''}, []);
  otherwise
    error('### Unknown file extension.');
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ssmFromMiir

PURPOSE ^

%%%

```
% 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 :    '$Id:$';
%
%%%
```

ssmFromPzmodel

PURPOSE ^

**%%%

```
%
% 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 $';
%
%%%
```

```
  ssmout(i_pzms).addHistory(ssm.getInfo('ssm', 'From Pzmodel'), plist(), {''}, pzmodelsin(i_pzms).hist);
end


end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ssmFromRepository

PURPOSE ^

**%%%

```
% PARAMETER:
%
% HISTORY:     15-04-2008 Hewitson
%              Creation
%
% VERSION : '$Id: ssmFromRepository.m,v 1.12 2008/08/08 15:32:33 adrien Exp $';
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ss = ssmFromRepository(pli)
utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

% Add default values
pl = combine(pli, ssm.getDefaultPlist('From Repository'));

% Get parameters
conn     = find(pl, 'conn');
hostname = find(pl, 'hostname');
database = find(pl, 'database');
ids      = find(pl, 'id');

% do we have a connection?
closeConn = 0;
if isempty(conn)
  closeConn = 1;
  % Connect to repository
  conn = mysql_connect(hostname, database);
end
if ~isa(conn, 'database')
  error('### connection failed.');
end

% Get each ID
Nids = length(ids);
ss  = [];
for kk=1:Nids

  %---- This id
  id = ids(kk);
  disp(sprintf('  - retrieving ID %d', id));

  %---- check ID object type
  tt = mysql_getObjType(conn, id);
  %---- If this is an AO
  if strcmp(tt, mfilename)
    %---- call database constructor
    s = ltpda_obj_retrieve(conn, id);
    %---- Add history
    s.addHistory(ssm.getInfo('ssm', 'From Repository'), pli, {''}, []);
    %---- Add to output array
    ss = [ss s]; %#ok<AGROW>
  else
    warning('    !skipping ID %d, type %s', id, tt);
  end

end

% close connection
if closeConn
  close(conn);
end
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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

    % Update here the structure

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (ssm) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

validate

PURPOSE ^

**%%%

SOURCE CODE ^

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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...?
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = validate(varargin)
  %% starting initial checks
  utils.helper.msg(utils.const.msg.MNAME, ['running ', mfilename]);

  % Collect all SSMs
  sys = utils.helper.collect_objects(varargin(:), 'ssm');

  % Decide on a deep copy or a modify, depending on the output
  sys = copy(sys, nargout);

  %% begin function body

  for i_sys = 1:length(sys) % going through the input

    %% =========== checking no compulsory field is missing ===========

    % calculating fields length for Ninputs, Inputsizes, sssize, outputsizes,
    % Nparams and checking compulsory fields are well defined

    %% =========== generating counter fields ===========
    sys(i_sys).Ninputs  = size(sys(i_sys).bmats, 2);
    sys(i_sys).Nss      = size(sys(i_sys).amats, 2);
    sys(i_sys).Noutputs = size(sys(i_sys).cmats, 1);
    sys(i_sys).Nparams  = length(sys(i_sys).paramnames);
    % generating inputsizes if it does not already exist
    if isempty(sys(i_sys).inputsizes)
      sys(i_sys).inputsizes = - ones(1,sys(i_sys).Ninputs);
      %  has to look in every b or d matrix to find a non-empty cell
      for i=1:sys(i_sys).Ninputs
        for j=1:sys(i_sys).Nss
          if ~isequal(sys(i_sys).bmats{j,i}, [])
            sys(i_sys).inputsizes(i) = size(sys(i_sys).bmats{j,i},2);
            break
          end
        end
        for j=1:sys(i_sys).Noutputs
          if ~isequal(sys(i_sys).dmats{j,i}, [])
            sys(i_sys).inputsizes(i) = size(sys(i_sys).dmats{j,i},2);
            break
          end
        end
      end
    end
```

```
    % generating sssize
    if isempty(sys(i_sys).sssizes)
      sys(i_sys).sssizes = - ones(1,sys(i_sys).Nss);
      %  has to look in every b or d matrix to find a non-empty cell
      for i=1:sys(i_sys).Nss
        sys(i_sys).sssizes(i) = size(sys(i_sys).amats{i,i},2); % to modify later on
      end
    end
    % generating outputsizes if it does not already exist
    if isempty(sys(i_sys).outputsizes)
      sys(i_sys).outputsizes = - ones(1,sys(i_sys).Noutputs);
      %  has to look in every b or d matrix to find a non-empty cell
      for i=1:sys(i_sys).Noutputs
        for j=1:sys(i_sys).Nss
          if ~isequal(sys(i_sys).cmats{i,j}, [])
            sys(i_sys).outputsizes(i) = size(sys(i_sys).cmats{i,j},1);
            break
          end
        end
        for j=1:sys(i_sys).Ninputs
          if ~isequal(sys(i_sys).dmats{i,j}, [])
            sys(i_sys).outputsizes(i) = size(sys(i_sys).dmats{i,j},1);
            break
          end
        end
      end
    end

    %% =========== generating missing user defined fields ===========

    % data with linear system matrices
    % generating mmats if it does not already exist
    if isempty(sys(i_sys).mmats)
      sys(i_sys).mmats = cell(sys(i_sys).Nss,sys(i_sys).Nss);
      for i=1:sys(i_sys).Nss
        sys(i_sys).mmats{i,i} = eye(sys(i_sys).sssizes(i));
      end
    end
    % generating isnumerical
    sys(i_sys).isnumerical = true;
    for i=1:length(sys(i_sys).amats)
      if ~isreal(sys(i_sys).amats{i})
        sys(i_sys).isnumerical = false;
      end
    end
    if sys(i_sys).isnumerical
      for i=1:length(sys(i_sys).bmats)
        if ~isreal(sys(i_sys).bmats{i})
          sys(i_sys).isnumerical = false;
        end
      end
    end
    if sys(i_sys).isnumerical
      for i=1:length(sys(i_sys).cmats)
        if ~isreal(sys(i_sys).cmats{i})
          sys(i_sys).isnumerical = false;
        end
      end
    end
    if sys(i_sys).isnumerical
      for i=1:length(sys(i_sys).dmats)
        if ~isreal(sys(i_sys).dmats{i})
          sys(i_sys).isnumerical = false;
        end
      end
    end
```

```
    if sys(i_sys).isnumerical
      for i=1:length(sys(i_sys).mmats)
        if ~isreal(sys(i_sys).mmats{i})
          sys(i_sys).isnumerical = false;
        end
      end
    end

    % nonlinear handle fields
    %generating amats_handles if it does not already exist
    if isempty(sys(i_sys).amats_handles)
      sys(i_sys).amats_handles = cell(sys(i_sys).Nss,sys(i_sys).Nss);
    end
    %generating bmats_handles if it does not already exist
    if isempty(sys(i_sys).bmats_handles)
      sys(i_sys).bmats_handles =cell(sys(i_sys).Nss,sys(i_sys).Ninputs);
    end
    %generating cmats_handles if it does not already exist
    if isempty(sys(i_sys).cmats_handles)
      sys(i_sys).cmats_handles = cell(sys(i_sys).Noutputs,sys(i_sys).Nss);
    end
    %generating dmats_handles if it does not already exist
    if isempty(sys(i_sys).dmats_handles)
      sys(i_sys).dmats_handles = cell(sys(i_sys).Noutputs,sys(i_sys).Ninputs);
    end

    % input variables array data
    % generating inputnames if it does not already exist
    if isempty(sys(i_sys).inputnames)
      sys(i_sys).inputnames = cell(1,sys(i_sys).Ninputs);
      for i=1:sys(i_sys).Ninputs
        sys(i_sys).inputnames{i} = ['input ',utils.helper.num2str(i)];
      end
    end
    % generating inputvarnames if it does not already exist
    if isempty(sys(i_sys).inputvarnames)
      sys(i_sys).inputvarnames = cell(1,sys(i_sys).Ninputs);
      for i=1:sys(i_sys).Ninputs
        sys(i_sys).inputvarnames{i} = cell(1,sys(i_sys).inputsizes(i));
        for j=1:sys(i_sys).inputsizes(i)
          sys(i_sys).inputvarnames{i}{j} = [sys(i_sys).inputnames{i},'>' utils.helper.num2str(j)];
        end
      end
    end

    % state space variables array data
    % generating ssnames if it does not already exist
    if isempty(sys(i_sys).ssnames)
      sys(i_sys).ssnames = cell(1,sys(i_sys).Nss);
      for i=1:sys(i_sys).Nss
        sys(i_sys).ssnames{i} = ['state ',utils.helper.num2str(i)];
      end
    end
    % generating ssvarnames if it does not already exist
    if isempty(sys(i_sys).ssvarnames)
      sys(i_sys).ssvarnames = cell(1,sys(i_sys).Nss);
      for i=1:sys(i_sys).Nss
        sys(i_sys).ssvarnames{i} = cell(1,sys(i_sys).sssizes(i));
        for j=1:sys(i_sys).sssizes(i)
          sys(i_sys).ssvarnames{i}{j} = [sys(i_sys).ssnames{i},'>' utils.helper.num2str(j)];
        end
      end
    end

    %for outputs
    % generating outputnames if it does not already exist
```

```
    if isempty(sys(i_sys).outputnames)
      sys(i_sys).outputnames = cell(1,sys(i_sys).Noutputs);
      for i=1:sys(i_sys).Noutputs
        sys(i_sys).outputnames{i} = ['output ',utils.helper.num2str(i)];
      end
    end
    % generating outputvarnames if it does not already exist
    if isempty(sys(i_sys).outputvarnames)
      sys(i_sys).outputvarnames = cell(1,sys(i_sys).Noutputs);
      for i=1:sys(i_sys).Noutputs
        sys(i_sys).outputvarnames{i} = cell(1,sys(i_sys).outputsizes(i));
        for j=1:sys(i_sys).outputsizes(i)
          sys(i_sys).outputvarnames{i}{j} = [sys(i_sys).outputnames{i},'>' utils.helper.num2str(j)];
        end
      end
    end

    % parameter fields
    % generating paramsigmas if it does not already exist
    if isempty(sys(i_sys).paramsigmas)
      sys(i_sys).paramsigmas = zeros(1,sys(i_sys).Nparams);
    end

    % generating paramvalues if it does not already exist
    if isempty(sys(i_sys).paramvalues)
      sys(i_sys).paramvalues = zeros(1,sys(i_sys).Nparams);
    end


    %% =========== checking field sizes ===========
    % checking compatibility with Ninputs
    if ~(    sys(i_sys).Ninputs==size(sys(i_sys).bmats,2)  ...
        && sys(i_sys).Ninputs==size(sys(i_sys).dmats,2) ...
        && sys(i_sys).Ninputs==size(sys(i_sys).bmats_handles,2)  ...
        && sys(i_sys).Ninputs==size(sys(i_sys).dmats_handles,2) ...
        && sys(i_sys).Ninputs==length(sys(i_sys).inputnames) ...
        && sys(i_sys).Ninputs==length(sys(i_sys).inputvarnames) ...
        && sys(i_sys).Ninputs==length(sys(i_sys).inputsizes) )
      error(['error in ssm ',sys(i_sys).name,' because there are ',...
        num2str(size(sys(i_sys).bmats,2)),' columns in B , ',...
        num2str(size(sys(i_sys).dmats,2)),' columns in D , ',...
        num2str(size(sys(i_sys).bmats_handles,2)),' columns in bmats_handles , ',...
        num2str(size(sys(i_sys).dmats_handles,2)),' columns in dmats_handles , ',...
        num2str(length(sys(i_sys).inputnames)),' inputnames, ',...
        num2str(length(sys(i_sys).inputvarnames)),' blocks in inputvarnames and ',...
        num2str(length(sys(i_sys).inputsizes)),' inputsizes'] );
    end

    % checking compatibility with Nss
    if ~(    sys(i_sys).Nss==size(sys(i_sys).amats,2)  ...
        && sys(i_sys).Nss==size(sys(i_sys).amats,1)  ...
        && sys(i_sys).Nss==size(sys(i_sys).bmats,1)  ...
        && sys(i_sys).Nss==size(sys(i_sys).cmats,2)  ...
        && sys(i_sys).Nss==size(sys(i_sys).amats_handles,2)  ...
        && sys(i_sys).Nss==size(sys(i_sys).amats_handles,1)  ...
        && sys(i_sys).Nss==size(sys(i_sys).bmats_handles,1)  ...
        && sys(i_sys).Nss==size(sys(i_sys).cmats_handles,2)  ...
        && sys(i_sys).Nss==length(sys(i_sys).ssnames) ...
        && sys(i_sys).Nss==length(sys(i_sys).ssvarnames) ...
        && sys(i_sys).Nss==length(sys(i_sys).sssizes) )
      error(['error in ssm ',sys(i_sys).name,' because there are ',...
        num2str(size(sys(i_sys).amats,2)),' columns in A, ',...
        num2str(size(sys(i_sys).amats,1)),' lines in A, ',...
        num2str(size(sys(i_sys).bmats,1)),' lines in B, ',...
        num2str(size(sys(i_sys).cmats,2)),' columns in C, ',...
        num2str(size(sys(i_sys).amats_handles,2)),' columns in A_handles, ',...
```

```
        num2str(size(sys(i_sys).amats_handles,1)),' lines in A_handles, ',...
        num2str(size(sys(i_sys).bmats_handles,1)),' lines in B_handles, ',...
        num2str(size(sys(i_sys).cmats_handles,2)),' columns in C_handles, ',...
        num2str(length(sys(i_sys).ssnames)),' ssnames, ',...
        num2str(length(sys(i_sys).ssvarnames)),' blocks in ssvarnames and ',...
        num2str(length(sys(i_sys).sssizes)),' sssizes'] );
    end

    % checking compatibility with Noutputs
    if ~(      sys(i_sys).Noutputs==size(sys(i_sys).cmats,1) ...
        && sys(i_sys).Noutputs==size(sys(i_sys).dmats,1) ...
        && sys(i_sys).Noutputs==size(sys(i_sys).cmats_handles,1) ...
        && sys(i_sys).Noutputs==size(sys(i_sys).dmats_handles,1) ...
        && sys(i_sys).Noutputs==length(sys(i_sys).outputnames) ...
        && sys(i_sys).Noutputs==length(sys(i_sys).outputvarnames) ...
        && sys(i_sys).Noutputs==length(sys(i_sys).outputsizes) )
      error(['error in ssm ',sys(i_sys).name,' because there are ',...
        num2str(size(sys(i_sys).cmats,1)),' lines in C, ',...
        num2str(size(sys(i_sys).dmats,1)),' lines in D, ',...
        num2str(size(sys(i_sys).cmats_handles,1)),' lines in C_handles, ',...
        num2str(size(sys(i_sys).dmats_handles,1)),' lines in D_handles, ',...
        num2str(length(sys(i_sys).outputnames)),' outputnames, ',...
        num2str(length(sys(i_sys).outputvarnames)),' blocks in outputvarnames and ',...
        num2str(length(sys(i_sys).outputsizes)),' outputsizes'] );
    end

    % Checking compatibility with Inputsizes
    for i=1:sys(i_sys).Ninputs
      if ~isequal(sys(i_sys).inputsizes(i), -1)
        if ~(  sys(i_sys).inputsizes(i) == length(sys(i_sys).inputvarnames{i}) )
          error(['error in ssm ', sys(i_sys).name, ...
            ' because the input number ', num2str(i),...
            ' named ', sys(i_sys).inputnames{i},  ...
            ' and of size ', num2str(sys(i_sys).inputsizes(i)), ...
            ' has an inputvarnames vector of size ', num2str(length(sys(i_sys).inputvarnames{i})) ]);
        end
      else
        str=['warning, input named ',sys(i_sys).inputnames{i},...
          ' has all matrices empty, should be deleted'] ;
        utils.helper.msg(utils.const.msg.MNAME,str);
      end
      for j=1:sys(i_sys).Noutputs
        if ~isequal(sys(i_sys).dmats{j,i}, [])
          if ~( sys(i_sys).inputsizes(i) == size(sys(i_sys).dmats{j,i},2) )
            error(['error in ssm ', sys(i_sys).name, ...
              ' because the input number ', num2str(i),...
              ' named ', sys(i_sys).inputnames{i},  ...
              ' and of size ', num2str(sys(i_sys).inputsizes(i)), ...
              ' has a D matrix of width ', num2str(size(sys(i_sys).dmats{j,i},2)) ]);
          end
        end
      end
      for j=1:sys(i_sys).Nss
        if ~isequal(sys(i_sys).bmats{j,i}, [])
          if ~( sys(i_sys).inputsizes(i) == size(sys(i_sys).bmats{j,i},2) )
            error(['error in ssm ', sys(i_sys).name, ...
              ' because the input number ', num2str(i),...
              ' named ', sys(i_sys).inputnames{i},  ...
              ' and of size ', num2str(sys(i_sys).inputsizes(i)), ...
              ' has a B matrix of width ', num2str(size(sys(i_sys).bmats{j,i},2)) ]);
          end
        end
      end
    end

    % Checking compatibility with Outputsizes
```

```
    for i=1:sys(i_sys).Noutputs
      if ~isequal(sys(i_sys).outputsizes(i), -1)
        if ~( sys(i_sys).outputsizes(i) == length(sys(i_sys).outputvarnames{i}) )
          error(['error in ssm ', sys(i_sys).name, ...
            ' because the output number ', num2str(i),...
            ' named ', sys(i_sys).outputnames{i},  ...
            ' and of size ', num2str(sys(i_sys).outputsizes(i)), ...
            ' has an outputvarnames vector of size ', num2str(length(sys(i_sys).outputvarnames{i})) ]);
        end
      else
        str = ['warning, output named ',sys(i_sys).outputnames{i},...
          ' has all matrices empty, should be deleted'];
        utils.helper.msg(utils.const.msg.MNAME, str);
      end
      for j=1:sys(i_sys).Ninputs
        if ~isequal(sys(i_sys).dmats{i,j}, [])
          if ~( sys(i_sys).outputsizes(i) == size(sys(i_sys).dmats{i,j},1) )
            error(['error in ssm ', sys(i_sys).name, ...
              ' because the output number ', num2str(i),...
              ' named ', sys(i_sys).outputnames{i},  ...
              ' and of size ', num2str(sys(i_sys).outputsizes(i)), ...
              ' has a D matrix of height ', num2str(size(sys(i_sys).dmats{i,j},1)) ]);
          end
        end
      end
      for j=1:sys(i_sys).Nss
        if ~isequal(sys(i_sys).cmats{i,j}, [])
          if ~( sys(i_sys).outputsizes(i) == size(sys(i_sys).cmats{i,j},1) )
            error(['error in ssm ', sys(i_sys).name, ...
              ' because the output number ', num2str(i),...
              ' named ', sys(i_sys).outputnames{i},  ...
              ' and of size ', num2str(sys(i_sys).outputsizes(i)), ...
              ' has a C matrix of height ', num2str(size(sys(i_sys).cmats{i,j},1)) ]);
          end
        end
      end
    end

    % Checking compatibility with sssizes
    for i=1:sys(i_sys).Nss
      if ~isequal(sys(i_sys).sssizes(i), -1)
        if ~( sys(i_sys).sssizes(i) == length(sys(i_sys).ssvarnames{i}) )
          error(['error in ssm ', sys(i_sys).name, ...
            ' because the state space number ', num2str(i),...
            ' named ', sys(i_sys).ssnames{i},  ...
            ' and of size ', num2str(sys(i_sys).sssizes(i)), ...
            ' has a ssvarnames vector of size ', num2str(length(sys(i_sys).ssvarnames{i})) ]);
        end
      else
        str = ['warning, output named ',sys(i_sys).outputnames{i},...
          ' has all matrices empty, should be deleted'];
        utils.helper.msg(utils.const.msg.MNAME, str);
      end
      for j=1:sys(i_sys).Ninputs
        if ~isequal(sys(i_sys).bmats{i,j}, [])
          if ~( sys(i_sys).sssizes(i) == size(sys(i_sys).bmats{i,j},1) )
            error(['error in ssm ', sys(i_sys).name, ...
              ' because the state space number ', num2str(i),...
              ' named ', sys(i_sys).ssnames{i},  ...
              ' and of size ', num2str(sys(i_sys).sssizes(i)), ...
              ' has a B matrix of height ', num2str(size(sys(i_sys).bmats{i,j},1)) ]);
          end
        end
      end
      for j=1:sys(i_sys).Nss
        if ~isequal(sys(i_sys).mmats{i,j}, [])
```

```
          if ~(  sys(i_sys).sssizes(i) == size(sys(i_sys).mmats{i,j},1) ...
              && sys(i_sys).sssizes(j) == size(sys(i_sys).mmats{i,j},2) )
            error(['error in ssm ', sys(i_sys).name, ...
              ' because the state space position ', num2str(i), ' ' , num2str(j),...
              ' named ', sys(i_sys).ssnames{i}, 'and ', sys(i_sys).ssnames{j},  ...
              ' and of size ', num2str([sys(i_sys).sssizes(i),sys(i_sys).sssizes(j)] ), ...
              ' has a A matrix of size ', num2str(size(sys(i_sys).amats{i,j})) ]);
          end
        end
      end
      for j=1:sys(i_sys).Noutputs
        if ~isequal(sys(i_sys).cmats{j,i}, [])
          if ~( sys(i_sys).sssizes(i) == size(sys(i_sys).cmats{j,i},2) )
            error(['error in ssm ', sys(i_sys).name, ...
              ' because the state space number ', num2str(i),...
              ' named ', sys(i_sys).ssnames{i},  ...
              ' and of size ', num2str(sys(i_sys).sssizes(i)), ...
              ' has a C matrix of width ', num2str(size(sys(i_sys).cmats{j,i},2)) ]);
          end
        end
      end
    end

    % checking compatibility with Nparams
    if ~(    sys(i_sys).Nparams == length(sys(i_sys).paramvalues) ...
        && sys(i_sys).Nparams == length(sys(i_sys).paramsigmas))
      error(['error in ssm ',sys(i_sys).name,' because there are ',...
        num2str(sys(i_sys).Nparams),' parameter names but ',...
        num2str(length(sys(i_sys).paramvalues)),' parameter values, and ',...
        num2str(length(sys(i_sys).paramsigmas)),' parametersigmas.']);
    end

    %% =========== checking redundancies ===========
    for i_input=1:sys(i_sys).Ninputs
      [pos, sum] = ssm.cellstrfind(sys(i_sys).inputnames,sys(i_sys).inputnames{i_input},'all');
      if sum>1
        error(['error because system ',sys(i_sys).name,' has ',num2str(sum),' inputs named ',sys(i_sys).inputnames{i_input}]);
      end
    end
    for i_output=1:sys(i_sys).Noutputs
      [pos, sum] = ssm.cellstrfind(sys(i_sys).outputnames,sys(i_sys).outputnames{i_output},'all');
      if sum>1
        error(['error because system ',sys(i_sys).name,' has ',num2str(sum),' outputs named ',sys(i_sys).outputnames{i_output}]);
      end
    end
    for i_ss=1:sys(i_sys).Nss
      [pos, sum] = ssm.cellstrfind(sys(i_sys).ssnames,sys(i_sys).ssnames{i_ss},'all');
      if sum>1
        error(['error because system ',sys(i_sys).name,' has ',num2str(sum),' state space named ',sys(i_sys).ssnames{i_ss}]);
      end
    end
    for i_input=1:sys(i_sys).Nparams
      if i_input <= sys(i_sys).Nparams
        [pos, sum] = ssm.cellstrfind(sys(i_sys).paramnames,sys(i_sys).paramnames{i_input},'all');
        if sum>1
          if sys(i_sys).paramvalues(pos(1))==sys(i_sys).paramvalues(pos(1))
            if sys(i_sys).paramsigmas(pos(1))==sys(i_sys).paramsigmas(pos(1))
              sys(i_sys).paramsigmas = [sys(i_sys).paramsigmas(1:pos(1)-1) sys(i_sys).paramsigmas(pos(1)+1 : sys(i_sys).Nparams)];
              sys(i_sys).paramvalues = [sys(i_sys).paramvalues(1:pos(1)-1) sys(i_sys).paramvalues(pos(1)+1 : sys(i_sys).Nparams)];
              sys(i_sys).paramnames = [sys(i_sys).paramnames(1:pos(1)-1) sys(i_sys).paramnames(pos(1)+1 : sys(i_sys).Nparams)];
              sys(i_sys).Nparams = sys(i_sys).Nparams-1;
              str = ['warning because parameter ', sys(i_sys).paramnames{pos(2)-1}, ' was found ', num2str(sum), ' times in ',
sys(i_sys).name];
              utils.helper.msg(utils.const.msg.MNAME, str);
            else
              str = ['error because system ',sys(i_sys).name,' has ',num2str(sum),' parameters named
```

```
',sys(i_sys).paramnames{i_input}, ' with different sigmas'];
                utils.helper.msg(utils.const.msg.MNAME, str);
              end
            else
              str = ['error because system ',sys(i_sys).name,' has ',num2str(sum),' parameters named ',sys(i_sys).paramnames{i_input},
' with different values'];
              utils.helper.msg(utils.const.msg.MNAME, str);
            end
          end
        end
      end

      %% =========== checking field types ===========

      Fields = {...
        'amats' 'bmats' 'cmats' 'dmats' 'mmats' 'isnumerical' 'timestep'...
        'amats_handles' 'bmats_handles' 'cmats_handles' 'dmats_handles' ...
        'inputnames' 'inputvarnames' 'inputsizes'  'Ninputs' ...
        'ssnames' 'ssvarnames' 'sssizes' 'Nss'  ...
        'outputnames' 'outputvarnames' 'outputsizes'  'Noutputs'...
        'paramnames' , 'paramvalues'  'paramsigmas' 'Nparams' };
      Classes = {...
        'doublesym' 'doublesym' 'doublesym' 'doublesym' 'doublesym' 'logical' 'double'...
        'cell' 'cell' 'cell' 'cell'...
        'cellstr' 'cellcellstr' 'double' 'double' ...
        'cellstr' 'cellcellstr' 'double' 'double'  ...
        'cellstr' 'cellcellstr' 'double' 'double' ...
        'cellstr' 'double' 'double' 'double' ...
        };

      for f = [Fields ; Classes]
        if isequal(f{2},'doublesym') % case where both double and symbolic classes are allowed
          for i_input = 1:length(sys(i_sys).(f{1}))
            if ~( isa(sys(i_sys).(f{1}){i_input},'double') || isa(sys(i_sys).(f{1}){i_input},'sym') )
              error(['error because in ssm ',sys(i_sys).name,' because element ',num2str(i_input),...
                ' of field ', f{1},' is of type ',class(sys(i_sys).(f{1})),...
                ' instead of '' double'' or ''sym'' ']);
            end
          end
        elseif isequal(f{2},'cellstr')
          if ~( iscellstr(sys(i_sys).(f{1})) )
            error(['error because in ssm ',sys(i_sys).name,...
              ' because  field ', f{1},' is of type ',class(sys(i_sys).(f{1})),...
              ' instead of ''cellstr'' ']);
          end
        elseif isequal(f{2},'cellcellstr')
          if ~( iscell(sys(i_sys).(f{1})) )
            error(['error because in ssm ',sys(i_sys).name,...
              ' because  field ', f{1},' is of type ',class(sys(i_sys).(f{1})),...
              ' instead of ''cellcellstr'' ']);
          end
          for i=1:length(sys(i_sys).(f{1}))
            if ~( iscellstr(sys(i_sys).(f{1}){i}) )
              error(['error because in ssm ',sys(i_sys).name,...
                ' because  field ', f{1},' is of type ',class(sys(i_sys).(f{1})),...
                ' instead of ''cellcellstr'' ']);
            end
          end
        else
          if ~isa(sys(i_sys).(f{1}),f{2})
            error(['error because in ssm ',sys(i_sys).name,' because field ',...
              f{1},' is of type ',class(sys(i_sys).(f{1})),' instead of ', f{2}]);
          end
        end
      end
    end
```

```
  if nargout > 0
    varargout = {sys};
  end
end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ssmFromss

PURPOSE ^

**%%%

```
0007 %
0008 % CALL:         see ssm
0009 %
0010 % PARAMETER:
0011 %
0012 % HISTORY:      15-04-2008 Adrien
0013 %               Creation
0014 %
0015 % TODO:         inplement 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 %%%%
```

fromA

PURPOSE ^

Construct an mfir from coefficients

SYNOPSIS ^

function f = fromA(f, pli)

DESCRIPTION ^

```
 Construct an mfir from coefficients
%%%
```

```
  a  = find(pl, 'a');
  fs = find(pl, 'fs');

  % Checking the coefficients are listed in rows
  if size(a,1)~=1
    a = a';
  end

  % Zero pad coefficients to avoid zero length history
  if numel(a) <= 1
    a = [a 0];
  end

  f.setName('A', 'internal');
  f.setFs(fs);
  f.setA(a);
  f.setGd((f.ntaps+1)/2);
  f.setHistout(zeros(1,f.ntaps-1));       % initialise output history

  % Add history
  f.addHistory(ii, pl, [], []);

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
%%%
```

```
ii.setMversion([VERSION '-->' ii.mversion]);

% Add default values
pl = combine(pli, ii.plists);

% Get parameters
a      = find(pl, 'AO');
N      = find(pl, 'N');
win    = find(pl, 'Win');
method = find(pl, 'method');

% Check that a.data is a fsdata object
if ~isa(a.data, 'fsdata')
  error('### Please use an analysis object with a fsdata data object to create a mfir object.');
end

fs = a.data.fs;
f  = a.data.getX;
xx = abs(a.data.getY);

ffm = f/(fs/2);
switch method
  case 'frequency-sampling'
    % check window
    if length(win.win) ~= N+1
      warning('!!! resizing window function to match desired filter order.');
      if strcmp(win.type, 'Kaiser') || strcmp(win.type, 'Flattop')
        win = specwin(win.type, N+1, win.psll);
      else
        win = specwin(win.type, N+1);
      end
    end
    utils.helper.msg(msg.OPROC2, 'designing filter using frequency-sampling method [help fir2]');
    mtaps = fir2(N, ffm, xx, win.win);
  case 'least-squares'
    error('### this design method is not working properly yet.');
    if mod(length(ffm),2)
      ffm = ffm(1:end-1);
      xx  = xx(1:end-1);
    end
    utils.helper.msg(msg.OPROC2, 'designing filter using least-squares method [help firls]');
    mtaps = firls(N, ffm, xx);
  case 'Parks-McClellan'
    error('### this design method is not working properly yet.');
    utils.helper.msg(msg.OPROC2, 'designing filter using Parks-McClellan method [help firpm]');
    mtaps = firpm(N, ffm, xx);
  otherwise
    error('### unknown filter design method.');
end

% Make mfir object
filt.setName(sprintf('fir(%s)', a.name), 'internal');
filt.setFs(fs);
filt.setA(mtaps);
filt.setGd((filt.ntaps+1)/2);
filt.setHistout(zeros(1,filt.ntaps-1));

% Add history
filt.addHistory(ii, pl, [], []);

end % function f = mfirFromAO(a, pli, version, algoname)
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromPzmodel

PURPOSE ^

Construct an mfir from a pzmodel

SYNOPSIS ^

function f = fromPzmodel(f, pli)

DESCRIPTION ^

```
 Construct an mfir from a pzmodel
%%%
```

```
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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromStandard

PURPOSE ^

Construct an mfir from a standard types

SYNOPSIS ^

function f = fromStandard(f, pli)

DESCRIPTION ^

```
 Construct an mfir from a standard types
%%%
```

```
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)
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
end

f.name    = 'std bandpass';
f.fs      = fs;
f.a       = g.*fir1(order, 2.*fc/fs, 'bandpass', win.win);
f.gd      = (f.ntaps)/2;
f.histout = zeros(1,f.ntaps-1);   % initialise output history


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = mkbandreject(f, pl)

  g      = find(pl, 'gain');
  fc     = find(pl, 'fc');
  fs     = find(pl, 'fs');
  order  = find(pl, 'order');
  win    = find(pl, 'Win');

  if(fc > fs/2)
    error('fc must be < fs/2');
  end
  if(fc(1) > fc(2))
    error('fc(1) must be < fc(2)');
```

```
end

f.name    = 'std bandreject';
f.fs      = fs;
f.a       = g.*fir1(order, 2.*fc/fs, 'stop', win.win);
f.gd      = (f.ntaps)/2;
f.histout = zeros(1,f.ntaps-1);   % initialise output history


```

*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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = mkhighpass(f, pl)

  g     = find(pl, 'gain');
  fc    = find(pl, 'fc');
  fs    = find(pl, 'fs');
  order = find(pl, 'order');
  win   = find(pl, 'Win');

  if(fc(1) > fs/2)
    error('fc must be < fs/2');
  end


```

```
f.name     = 'std highpass';
f.fs       = fs;
f.a        = g.*fir1(order, 2*fc/fs, 'high', win.win);
f.gd       = (f.ntaps)/2;
f.histout  = zeros(1,f.ntaps-1);   % initialise output history
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

mklowpass

PURPOSE ^

MKLOWPASS return a low pass filter mfir().

SYNOPSIS ^

function f = mklowpass(f, pl)

DESCRIPTION ^

```
 MKLOWPASS return a low pass filter mfir().
%%%
```

```
f.gd        = (f.ntaps)/2;
f.histout   = zeros(1,f.ntaps-1);   % initialise output history


```

*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
standard FIR filter.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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]
```

```
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
%
%%%
```

setGd

PURPOSE ^

SETGD Set the property 'gd'

SYNOPSIS ^

function obj = setGd(obj, val)

DESCRIPTION ^

```
 SETGD Set the property 'gd'
%%%
```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

    if ~isempty(obj_struct.gain) && obj_struct.gain ~= 0
      obj_struct.a = obj_struct.a * obj_struct.gain;
      obj_struct = rmfield(obj_struct, 'gain');
    end

    if isempty(obj_struct.plist)
      obj_struct.plist = plist();
    end
    ii = minfo('update_struct', 'mfir', '', '', ...
               '$Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $', ...
               {'Default'}, plist(), 0, 0);
    obj_struct.hist = history(time().utc_epoch_milli, ii, obj_struct.plist);
    obj_struct = rmfield(obj_struct, 'plist');

    obj_struct.creator = provenance('Updater 1.0 -> 1.1');
    obj_struct.gd      = 0;

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (mfir) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef ltpda_filter < ltpda_uoh


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                          %
```

```
%%%
```

```
0139
0140    %%%%
```

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 ^

```
function f = filload(filename)

  % 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 $
  %

  delims = ' \n';
  in = fopen(filename, 'rt');

  name = filename(1:end-4);

  if(in < 0)
    error(['can not open file ' filename]);
  end
  lbuffer = fgets(in);
  n = 1;
  nacoeffs = 0;
  nbcoeffs = 0;
  filt.a = [];
  filt.b = [];
  norminst = '';

  while (lbuffer > 0)
    if(length(lbuffer) > 1)
      % read the first token on this line
      [token, r] = strtok (lbuffer, delims);

      % Get fs
      if(strcmp(token, 'iir'))
        [token, r] = strtok (r, delims); % value
        fs = str2double(token);
      end

      % Get a coeffs
```

```
      if(token(1) == 'a')
        [token, r] = strtok (r, delims); % value
        nacoeffs = nacoeffs+1;
        filt.a(nacoeffs) = str2double(token);
      end
      % Get b coeffs
      if(token(1) == 'b')
        [token, r] = strtok (r, delims); % value
        nbcoeffs = nbcoeffs+1;
        filt.b(nbcoeffs) = str2double(token);
      end
    end

    n = n + 1;
    lbuffer = fgets(in);
  end
  fclose(in);

  if(nacoeffs > nbcoeffs)
    error('## unstable filter: nacoeffs > nbcoeffs');
  end

  if(nacoeffs < nbcoeffs)
    filt.a = [filt.a zeros(1,(nbcoeffs-nacoeffs))];
  end

  ncoeffs = nbcoeffs;

  f.name    = name;
  f.fs      = fs;
  f.a       = filt.a;
  f.b       = filt.b;
  f.ntaps   = ncoeffs;
  f.gain    = 1;
  f.histin  = zeros(1, f.ntaps-1);
  f.histout = zeros(1, f.ntaps-1);
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromAB

PURPOSE ^

Construct an miir from coefficients

SYNOPSIS ^

function f = fromAB(f, pli)

DESCRIPTION ^

```
 Construct an miir from coefficients
%%%
```

```
  % Checking the coefficients are listed in rows
  if size(a,1)~=1
    a = a';
  end
  if size(b,1)~=1
    b = b';
  end

  % Zero pad to avoid 0 length history vector
  if numel(a) <= 1
    a = [a 0];
  end
  if numel(b) <= 1
    b = [b 0];
  end

  f.setName('AB', 'internal');
  f.setFs(fs);
  f.setA(a);
  f.setB(b);
  f.setHistin(zeros(1,f.ntaps-1));       % initialise input history
  f.setHistout(zeros(1,f.ntaps-1));       % initialise output history

  % Add history
  f.addHistory(ii, pl, [], []);

end % End fromAB
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
%%%
```

```
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
%%%
```

```
  fs  = find(pl, 'fs');

  if isempty(fs)
    % get max freq in pzmodel
    fs = 8*getupperFreq(pzm);
    warning([sprintf('!!! no sample rate specified. Designing for fs=%2.2f Hz.', fs)...
      sprintf('\nThe filter will be redesigned later when used.')]);
  end
  % make MIIR filter
  f = tomiir(pzm, fs);
  f.setName(pzm.name, 'internal');

  % Add history
  f.addHistory(ii, pl, [], pzm.hist);

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
%%%
```

```
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.plists);
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)
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
% Build filter coefficients
[a, b] = cheby1(order, ripple, 2.*fc./fs);

% Set filter properties
f.setName('bandpass', 'internal');
f.setFs(fs);
f.setA(g.*a);
f.setB(b);
f.setHistin(zeros(1,f.ntaps-1));   % initialise input history
f.setHistout(zeros(1,f.ntaps-1));   % initialise output history


```

*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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = mkbandreject(f, pl)

  g      = find(pl, 'gain');
  fc     = find(pl, 'fc');
  fs     = find(pl, 'fs');
  order  = find(pl, 'order');
  ripple = find(pl, 'ripple');

  if(fc > fs/2)
    error('fc must be < fs/2');
  end
  if(fc(1) > fc(2))
    error('fc(1) must be < fc(2)');
```

```
end

% Build filter coefficients
[a, b] = cheby1(order, ripple, 2.*fc./fs, 'stop');

% Set filter properties
f.setName('bandreject', 'internal');
f.setFs(fs);
f.setA(g.*a);
f.setB(b);
f.setHistin(zeros(1,f.ntaps-1));   % initialise input history
f.setHistout(zeros(1,f.ntaps-1));   % initialise output history
```

*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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = mkhighpass(f, pl)

  g     = find(pl, 'gain');
  fc    = find(pl, 'fc');
  fs    = find(pl, 'fs');
  order = find(pl, 'order');


  if(fc > fs/2)
    error('fc must be < fs/2');
  end

  % Build filter coefficients
```

```
[a, b] = butter(order, 2*fc/fs, 'high');

% Set filter properties
f.setName('highpass', 'internal');
f.setFs(fs);
f.setA(g.*a);
f.setB(b);
f.setHistin(zeros(1,f.ntaps-1));       % initialise input history
f.setHistout(zeros(1,f.ntaps-1));       % initialise output history


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function f = mklowpass(f, pl)

  g     = find(pl, 'gain');
  fc    = find(pl, 'fc');
  fs    = find(pl, 'fs');
  order = find(pl, 'order');

  if(fc > fs/2)
    error('fc must be < fs/2');
  end

  % Build coefficients
  [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
standard IIR filter.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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]
```

```
%               '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
%
%%%
```

setB

PURPOSE ^

SETB Set the property 'b'

SYNOPSIS ^

function obj = setB(obj, val)

DESCRIPTION ^

```
 SETB Set the property 'b'
%%%
```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

    if ~isempty(obj_struct.gain) && obj_struct.gain ~= 0
      obj_struct.a = obj_struct.a * obj_struct.gain;
      obj_struct = rmfield(obj_struct, 'gain');
    end

    if isempty(obj_struct.plist)
      obj_struct.plist = plist();
    end
    ii = minfo('update_struct', 'miir', '', '', ...
               '$Id: update_struct.m,v 1.1 2008/07/10 12:06:49 ingo Exp $', ...
               {'Default'}, plist(), 0, 0);
    obj_struct.hist = history(time().utc_epoch_milli, ii, obj_struct.plist);
    obj_struct = rmfield(obj_struct, 'plist');

    obj_struct.creator = provenance('Updater 1.0 -> 1.1');

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (miir) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

computeInterval

PURPOSE ^

COMPUTEINTERVAL compute the interval of the time span.

SYNOPSIS ^

function ts = computeInterval(ts)

DESCRIPTION ^

```
 COMPUTEINTERVAL compute the interval of the time span.
%%%
```

```
% VERSION:     $Id: computeInterval.m,v 1.1 2008/07/23 17:31:56 ingo Exp $
%
% HISTORY:     03-08-2007 Diepholz
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ts = computeInterval(ts)

  ts.interval = '';

  number = abs(ts.startT.utc_epoch_milli-ts.endT.utc_epoch_milli);

  if (ts.endT.utc_epoch_milli-ts.startT.utc_epoch_milli < 0)
    ts.interval = '-';
  end

  form = java.text.SimpleDateFormat;
  form.setTimeZone(java.util.TimeZone.getTimeZone('UTC'));

  form.applyPattern('yyyy')
  num_y = char(form.format(java.util.Date(number)));
  num_y = str2double(num_y);
  num_y = num_y - 1970;
  num_y = sprintf('%02d', num_y);

  form.applyLocalizedPattern('DD')
  num_d = char(form.format(java.util.Date(number)));
  num_d = sprintf('%02d', str2double(num_d)-1);

  form.applyLocalizedPattern('HH')
  num_h = char(form.format(java.util.Date(number)));

  form.applyLocalizedPattern('mm')
  num_m = char(form.format(java.util.Date(number)));

  form.applyLocalizedPattern('ss')
  num_s = char(form.format(java.util.Date(number)));

  form.applyLocalizedPattern('SSS')
  num_milli_s = char(form.format(java.util.Date(number)));

  if ~strcmp(num_y, '00')
    ts.interval = [ts.interval num_y ' Years '];
  end

  if ~strcmp(num_d, '00')
    ts.interval = [ts.interval num_d ' Days '];
  end

  if ~strcmp(num_h, '00')
    ts.interval = [ts.interval num_h ' Hours '];
  end

  if ~strcmp(num_m, '00')
    ts.interval = [ts.interval num_m ' Minutes '];
  end

  if ~strcmp(num_s, '00')
    ts.interval = [ts.interval num_s ' Seconds '];
  end

  if ~strcmp(num_milli_s, '000')
    ts.interval = [ts.interval num_milli_s ' Milliseconds '];
  end
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromTimespanDef

PURPOSE ^

**%%%

```
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 %%%%
```

```
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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

    % Update here the structure

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (time) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function time_str = epoch2str(tt)
  % 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
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
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);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

```
%               Creation
%
%%%
```

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.
%%%
```

```
%               Creation
%
%%%
```

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 varargout = parse_time_string(varargin)
% PARSE_TIME_STRING into

  %        'match expression'   'JAVA date format'  'MATLAB date format
  parse = {'\d{2}:\d{2}:\d{2}',    'HH:mm:ss',           'HH:MM:SS';    ...
           '\d{2}:\d{2}',          'mm:ss',              'MM:SS';       ...
           '\d{2}-\d{2}-\d{4}',    'dd-MM-yyyy',         'dd-mm-yyyy'; ...
           '\d{2}\.\d{2}\.\d{4}', 'dd.MM.yyyy',         'dd.mm.yyyy'; ...
           '\d{4}-\d{2}-\d{2}',    'yyyy-MM-dd',         'yyyy-mm-dd'; ...
           '\d{4}\.\d{2}\.\d{2}', 'yyyy.MM.dd',         'yyyy.mm.dd'; ...
           '\d{2}-\d{2}',          'MM-dd',              'mm-dd';       ...
           '\.\d{1,3}',            '.SSS',               '.FFF'};

  ret_java   = varargin{1};
  ret_matlab = varargin{1};

  for ii = 1:size(parse,1)

    curr_parse = parse{ii,1};
    [match] = regexp(ret_java, curr_parse, 'match');

    if ~isempty(match) && length(match) == 1
      replace_java   = parse{ii,2};
      replace_matlab = parse{ii,3};
      ret_java   = strrep(ret_java, match{1}, replace_java);
      ret_matlab = strrep(ret_matlab, match{1}, replace_matlab);
    end

  end

  % Set outputs
  varargout{1} = ret_java;
  varargout{2} = ret_matlab;

end
% 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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');
    obj_struct = rmfield(obj_struct, 'plist');

    if isstruct(obj_struct.timeformat)
      obj_struct.timeformat = obj_struct.timeformat.format_str;
    end

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (time) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fq2ri

PURPOSE ^

FQ2RI Convert frequency/Q pole/zero representation

SYNOPSIS ^

function ri= fq2ri(f0, Q)

DESCRIPTION ^

```
 FQ2RI Convert frequency/Q pole/zero representation
%%%
```

```
      G   = 1/(w0*w0);
      re  = G.*-w0/(2*Q);
      im  = G.*w0*sqrt(4*Q*Q-1)/(2*Q);
      tri = complex(re,im);
      ri  = [tri; conj(tri)];
    end
  end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ri2fq

PURPOSE ^

RI2FQ Convert comlpex pole/zero into frequency/Q pole/zero representation.

SYNOPSIS ^

function [f0, q]= ri2fq(c)

DESCRIPTION ^

```
 RI2FQ Convert comlpex pole/zero into frequency/Q pole/zero representation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 DESCRIPTION: RI2FQ Convert comlpex 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 comlpex pole/zero into frequency/Q pole/zero representation.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0003 %
0004 % DESCRIPTION: RI2FQ Convert comlpex 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;
```

```
q = sqrt(b^2/a^2 +1)/2;

% 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'
%%%
```

```
  ii.ri = pz.fq2ri(ii.f, ii.q);
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'
%%%
```

```
  ii.ri = pz.fq2ri(ii.f, ii.q);

  %%% did we split to two real poles?
  if ii.q < 0.5
    ri = ii.ri;
    ii.ri = ri(1);
    ii(2).f  = ii.f;
    ii(2).q  = NaN;
    ii(2).ri = ri(1);
  end
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'
%%%
```

```
  elseif numel(val) == 2
    if val(1) ~= conj(val(2))
      error('### Please enter a conjugate pair to specify a complex pole.');
    end
  else
    error('### A pole/zero must be defined with a conjugate pair or a single complex number');
  end

  %%% set 'ri'
  ii.ri = val;
  %%% Then compute and set f and Q
  [f,q] = pz.ri2fq(ii.ri);
  ii.f = f;
  ii.q = q;
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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');
    obj_struct = rmfield(obj_struct, 'plist');

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (pz) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;
```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (minfo) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.9.1'
  struct_ver = '1.9.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.1' ->'1.9.2'  %%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.9.2';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    %%% methodInfo
    ii = minfo(obj_struct.name, 'ao', '', '', obj_struct.version, {''}, plist, 0, 0);
    obj_struct.methodInfo = ii;

    %%% version
    obj_struct.version = obj_struct.consver;

    %%% proctime
    obj_struct.proctime = obj_struct.created;

    %%% methodInvars
    obj_struct.methodInvars = obj_struct.invars;

    %%% plistUsed
    obj_struct.plistUsed = obj_struct.plist;
  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (history) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver)

  try
    obj_struct.version = 'updater 1.0';
    obj_struct.symbolic_math_version = 'updateer 1.0';
  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (provenance) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.9.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try

    obj_struct = rmfield(obj_struct, 'name');

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (cdata) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.9.1'
  struct_ver = '1.9.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.1' ->'1.9.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.1')

  % Check Y units
  try
    uo = unit(obj_struct.yunits);
  catch
    warning('!!! This XML file contains a fsdata object with unsupported y-units [%s]. Setting to empty.', obj_struct.yunits);
    obj_struct.yunits = '';
  end

  % Check X units
  try
    uo = unit(obj_struct.xunits);
  catch
    warning('!!! This XML file contains a fsdata object with unsupported x-units [%s]. Setting to empty.', obj_struct.xunits);
    obj_struct.xunits = '';
  end

  % Set the struct_ver to '1.9.2'
  struct_ver = '1.9.2';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.2' ->'1.9.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.2')

  % Set the struct_ver to '1.9.3'
  struct_ver = '1.9.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function f = getFfromYFs(N,fs)
% 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
%
%

f = linspace(0, fs/2, N);


% 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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.9.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');
    obj_struct.t0 = time(0);

    if isempty(obj_struct.navs)
      obj_struct.navs = NaN;
    end

    if isempty(obj_struct.fs)
      obj_struct.fs = NaN;
    end

    if isempty(obj_struct.enbw)
      obj_struct.enbw = NaN;
    end


  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (fsdata) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.9.1'
  struct_ver = '1.9.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.1' ->'1.9.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.1')

  % Make sure we have strings for the units
  obj_struct.xunits = char(obj_struct.xunits);
  obj_struct.yunits = char(obj_struct.yunits);

  % We need to fix any strange old units

  %- /\surdHz -> Hz^-0.5
  obj_struct.xunits = strrep(obj_struct.xunits, '/\surdHz', ' Hz^-0.5');
  obj_struct.yunits = strrep(obj_struct.yunits, '/\surdHz', ' Hz^-0.5');

  % Check Y units
  try
    uo = unit(obj_struct.yunits);
  catch
    warning('!!! This XML file contains a fsdata object with unsupported y-units [%s]. Setting to empty.', obj_struct.yunits);
    obj_struct.yunits = '';
  end

  % Check X units
  try
    uo = unit(obj_struct.xunits);
  catch
    warning('!!! This XML file contains a fsdata object with unsupported x-units [%s]. Setting to empty.', obj_struct.xunits);
    obj_struct.xunits = '';
  end
  % Set the struct_ver to '1.9.2'
  struct_ver = '1.9.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.2' ->'1.9.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.2')

  % Set the struct_ver to '1.9.3'
  struct_ver = '1.9.3';

end

varargout{1} = obj_struct;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

- 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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef data2D < ltpda_data

```

```
%%%
```

```
      obj.y = val;

      sx = size(obj.x);
      sy = size(obj.y);
      if sx(1) == 1 && sy(1) ~= 1
        obj.x = obj.x.';
      end
      if sx(2) == 1 && sy(2) ~= 1
        obj.x = obj.x.';
      end

    end
  end

  %%%%
```

```
  %%%%
```

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 ^

```
% 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 $
%

function varargout = fitfs(varargin)

  import utils.const.*

  tol = 1e-2;

  % Get input vertices
  xi  = varargin{1};
  % reshape x to match with linspace output
  ss = size(xi);
  if ss(1) > ss(2)
    xi = xi.';
  end
  d = diff(xi);

  % Some statistics on x
  medd  = median(d);
  mind  = min(d);
  maxd  = max(d);
  % Initial guess at sample rate
  ifs   = 1.0/medd;

  % Do we need a fit?
  fitted = false;
  if abs(maxd-medd)/medd>tol || abs(mind-medd)/medd>tol

    utils.helper.msg(msg.OPROC1, 'fitting t0 and sample rate for unevenly sampled data.');

%     % We need sigma for fit constraints
%     sigma = std(1./d);
%
%     % Initial guess
%     x0 = [xi(1) ifs];
%
%     lb = [-1/ifs (1-sigma)*ifs];
%     ub = [1/ifs (1+sigma)*ifs];
%
%     %     'Display','iter',...
%     options = optimset('MaxFunEvals', 1e3, ...
%       'TolFun', 1e-20, 'TolX', 1e-20, 'MaxIter', 1e3);
%
%     x = lsqcurvefit(@tgrid, x0, xi, xi, lb, ub, options);

%     utils.helper.msg(msg.OPROC2, 'fit returned: %g s, %g Hz', x(1), x(2));
    % Return only 3 significant figures
%     x(1) = eval(sprintf('%0.8g', x(1)));
%     x(2) = eval(sprintf('%0.8g', x(2)));
    fitted = true;
    % We return the median and the intial sample time
    x(1) = 0; %eval(sprintf('%0.8g', xi(1)));
    x(2) = eval(sprintf('%0.8g', ifs));
    utils.helper.msg(msg.OPROC2, 'setting: %g s, %g Hz', x(1), x(2));
  else
    % We return the median and the intial sample time
    x(1) = eval(sprintf('%0.8g', xi(1)));
    x(2) = eval(sprintf('%0.8g', ifs));
  end


  % Set outputs
  if nargout == 0
    disp(x);
```

```
  elseif nargout == 1
    varargout{1} = x(2);
  elseif nargout == 2
    varargout{1} = x(2);
    varargout{2} = x(1);
  elseif nargout == 3
    varargout{1} = x(2);
    varargout{2} = x(1);
    varargout{3} = fitted;
  else
    error('### Unknown number of outputs.');
  end

  %-------------------------------
  % Target function to minimise
  %-------------------------------
  function t = tgrid(x, xdata)
    t0  = x(1);
    fs  = x(2);
    N = length(xdata);
    t = linspace(t0, (N-1)/fs, N);
  end

end % End fitfs
% 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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.9.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');


    if isempty(obj_struct.fs)
      obj_struct.fs = NaN;
    end

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (fsdata) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.9.1'
  struct_ver = '1.9.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.1' ->'1.9.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.1')


  % Check Y units
  try
    uo = unit(obj_struct.yunits);
  catch
    warning('!!! This XML file contains a fsdata object with unsupported y-units [%s]. Setting to empty.', obj_struct.yunits);
    obj_struct.yunits = '';
  end

  % Check X units
  try
    uo = unit(obj_struct.xunits);
  catch
    warning('!!! This XML file contains a fsdata object with unsupported x-units [%s]. Setting to empty.', obj_struct.xunits);
    obj_struct.xunits = '';
  end

  % Set the struct_ver to '1.9.2'
  struct_ver = '1.9.2';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.2' ->'1.9.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.2')

  % Set the struct_ver to '1.9.3'
  struct_ver = '1.9.3';

end

varargout{1} = obj_struct;
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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.9.1'   %%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');
  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (fsdata) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.9.1'
  struct_ver = '1.9.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.1' ->'1.9.2'  %%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.1')

    % Check Y units
    try
      uo = unit(obj_struct.yunits);
    catch
      warning('!!! This XML file contains a fsdata object with unsupported y-units [%s]. Setting to empty.', obj_struct.yunits);
      obj_struct.yunits = '';
    end

    % Check X units
    try
      uo = unit(obj_struct.xunits);
    catch
      warning('!!! This XML file contains a fsdata object with unsupported x-units [%s]. Setting to empty.', obj_struct.xunits);
      obj_struct.xunits = '';
    end

  % Set the struct_ver to '1.9.2'
  struct_ver = '1.9.2';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.2' ->'1.9.3'  %%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.2')

  % Set the struct_ver to '1.9.3'
  struct_ver = '1.9.3';

end

varargout{1} = obj_struct;

```

*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
%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.0')

  try
    obj_struct = rmfield(obj_struct, 'name');
    obj_struct = rmfield(obj_struct, 'created');
  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (fsdata) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.1' ->'1.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.1')

  % Set the struct_ver to '1.2'
  struct_ver = '1.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
%%%
```

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef data3D < data2D


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties
    zunits = unit;
    z      = [];
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end
```

```
%---------- Abstract Properties ----------
properties (Abstract = true, SetAccess = protected)
end

%%%
```

```
  end

  %%%%
```

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.
   <a href="matlab:help ao/diff">ao/diff</a>                 -  differentiates the data in AO.
   <a href="matlab:help ao/display">ao/display</a>              -  implement terminal display
for analysis object.
   <a href="matlab:help ao/dopplercorr">ao/dopplercorr</a>         - FOO description for
function 'foo' in one line. Necessary for lookfor functionality.
   <a href="matlab:help ao/downsample">ao/downsample</a>          -  AOs containing time-
series data.
   <a href="matlab:help ao/dropduplicates">ao/dropduplicates</a>      -  drops all duplicate
samples in time-series AOs.
   <a href="matlab:help ao/dsmean">ao/dsmean</a>               -  performs a simple
downsampling by taking the mean of every N samples.
   <a href="matlab:help ao/eig">ao/eig</a>                 -  overloads the determinant
function for Analysis objects.
   <a href="matlab:help ao/exp">ao/exp</a>                 -  overloads the exp operator for
Analysis objects. Exponential.
   <a href="matlab:help ao/export">ao/export</a>               -  export an analysis object to
a text file.
   <a href="matlab:help ao/extractm">ao/extractm</a>            -  extracts an m-file from an
analysis object and saves it to disk.
   <a href="matlab:help ao/extractmdl">ao/extractmdl</a>          -  extracts an mdl file
from an analysis object and saves it to disk.
   <a href="matlab:help ao/fft">ao/fft</a>                 -  overloads the fft method for
Analysis objects.
   <a href="matlab:help ao/filter">ao/filter</a>               -  overrides the filter
function for analysis objects.
   <a href="matlab:help ao/filtfilt">ao/filtfilt</a>            -  overrides the filtfilt
function for analysis objects.
   <a href="matlab:help ao/find">ao/find</a>                 -  particular samples that
satisfy the input query and return a new AO.
   <a href="matlab:help ao/findFsMax">ao/findFsMax</a>           -  Returns the max Fs of a
set of AOs
   <a href="matlab:help ao/findShortestVector">ao/findShortestVector</a>  -  Returns the
length of the shortest vector in samples
   <a href="matlab:help ao/firwhiten">ao/firwhiten</a>           -  whitens the input time-
series by building an FIR whitening filter.
   <a href="matlab:help ao/fixfs">ao/fixfs</a>               -  resamples the input time-
series to have a fixed sample rate.
   <a href="matlab:help ao/fngen">ao/fngen</a>               -  creates an arbitrarily long
time-series based on the input PSD.
   <a href="matlab:help ao/fq2fac">ao/fq2fac</a>               -  is a private function and is
called by ngconv.m which can be found in the
   <a href="matlab:help ao/fromDataInMAT">ao/fromDataInMAT</a>       -  Convert a saved data-
array into an AO with a tsdata-object
   <a href="matlab:help ao/fromDatafile">ao/fromDatafile</a>        -  Construct an ao from
filename AND parameter list
   <a href="matlab:help ao/fromFSfcn">ao/fromFSfcn</a>           -  Construct an ao from a
fs-function string
   <a href="matlab:help ao/fromFcn">ao/fromFcn</a>             -  Construct an ao from a
function string
   <a href="matlab:help ao/fromPolyval">ao/fromPolyval</a>         -  Construct a time-series
ao from polynomial coefficients
   <a href="matlab:help ao/fromPzmodel">ao/fromPzmodel</a>         -  Construct a time-series
ao from polynomial coefficients
   <a href="matlab:help ao/fromSpecWin">ao/fromSpecWin</a>         -  Construct an ao from a
Spectral window
   <a href="matlab:help ao/fromTSfcn">ao/fromTSfcn</a>           -  Construct an ao from a
ts-function string
   <a href="matlab:help ao/fromVals">ao/fromVals</a>            -  Construct an ao from a
value set
   <a href="matlab:help ao/fromWaveform">ao/fromWaveform</a>        -  Construct an ao from a
waveform
   <a href="matlab:help ao/fs">ao/fs</a>                  -  Get the data property 'fs'.
   <a href="matlab:help ao/gapfilling">ao/gapfilling</a>          -  fills possible gaps in
data.
   <a href="matlab:help ao/ge">ao/ge</a>                  -  overloads >= operator for
analysis objects. Compare the y-axis values.
   <a href="matlab:help ao/gt">ao/gt</a>                  -  overloads > operator for
analysis objects. Compare the y-axis values.
   <a href="matlab:help ao/hist">ao/hist</a>                -  overloads the histogram
function (hist) of MATLAB for Analysis Objects.
   <a href="matlab:help ao/ifft">ao/ifft</a>                -  overloads the ifft operator
for Analysis objects.
   <a href="matlab:help ao/imag">ao/imag</a>                -  overloads the imaginary
operator for Analysis objects.
   <a href="matlab:help ao/index">ao/index</a>               -  index into an AO array or
matrix. This properly captures the history.
   <a href="matlab:help ao/interp">ao/interp</a>               -  interpolate the values in
the input AO(s) at new values.
   <a href="matlab:help ao/interpmissing">ao/interpmissing</a>       -  interpolate missing
samples in a time-series.
   <a href="matlab:help ao/inv">ao/inv</a>                  -  overloads the inverse function
for Analysis Objects.
   <a href="matlab:help ao/iplot">ao/iplot</a>               -  provides an intelligent
plotting tool for LTPDA.
   <a href="matlab:help ao/join">ao/join</a>                -  multiple AOs into a single AO.
```

```
   <a href="matlab:help ao/lcohere">ao/lcohere</a>              -  implement coherence
estimation computed on a log frequency axis.
   <a href="matlab:help ao/lcpsd">ao/lcpsd</a>                 -  implement cross-power-
spectral density estimation computed on a log frequency axis.
   <a href="matlab:help ao/le">ao/le</a>                   -  overloads <= operator for
analysis objects. Compare the y-axis values.
   <a href="matlab:help ao/len">ao/len</a>                  -  overloads the length operator
for Analysis objects. Length of the data samples.
   <a href="matlab:help ao/lincom">ao/lincom</a>               - (No help available)
   <a href="matlab:help ao/linedetect">ao/linedetect</a>           -  find spectral lines in
the ao/fsdata objects.
   <a href="matlab:help ao/ln">ao/ln</a>                   -  overloads the log operator for
Analysis objects. Natural logarithm.
   <a href="matlab:help ao/log">ao/log</a>                  -  overloads the log operator for
Analysis objects. Natural logarithm.
   <a href="matlab:help ao/log10">ao/log10</a>                 -  overloads the log10 operator
for Analysis objects. Common (base 10) logarithm.
   <a href="matlab:help ao/lpsd">ao/lpsd</a>                 -  implement the LPSD algorithm
for analysis objects.
   <a href="matlab:help ao/lt">ao/lt</a>                   -  overloads < operator for
analysis objects. Compare the y-axis values.
   <a href="matlab:help ao/ltf_plan">ao/ltf_plan</a>             -  computes all input values
needed for the LPSD and LTFE algorithms.
   <a href="matlab:help ao/ltfe">ao/ltfe</a>                 -  implement transfer-function
estimation computed on a log frequency axis.
   <a href="matlab:help ao/lxspec">ao/lxspec</a>               -  performs log-scale cross-
spectral analysis of various forms.
   <a href="matlab:help ao/max">ao/max</a>                  -  computes the maximum value of
the data in the AO.
   <a href="matlab:help ao/mchol">ao/mchol</a>                - (No help available)
   <a href="matlab:help ao/md5">ao/md5</a>                  -  computes an MD5 checksum from
an analysis objects.
   <a href="matlab:help ao/mdc1_ifo2acc_fd">ao/mdc1_ifo2acc_fd</a>     - MDC1_IFO2ACC_FS
calculates the external acceleration in the frequency-domain.
   <a href="matlab:help ao/mdc1_ifo2acc_inloop">ao/mdc1_ifo2acc_inloop</a> -  calculates the
inloop acceleration in the time-domain.
   <a href="matlab:help ao/mdc1_ifo2control">ao/mdc1_ifo2control</a>    -  converts the input
time-series to control forces.
   <a href="matlab:help ao/mdc1_input_noises">ao/mdc1_input_noises</a>   - %
LTPDA_MDC1_IFO_NOISE returns a model spectrum of the IFO sensing noise for MDC1.
   <a href="matlab:help ao/mdc1_x2acc">ao/mdc1_x2acc</a>           -  converts the input time-
series to acceleration with a time-domain filter
   <a href="matlab:help ao/mean">ao/mean</a>                 -  computes the mean value of
the data in the AO.
   <a href="matlab:help ao/median">ao/median</a>               -  computes the median value of
the data in the AO.
   <a href="matlab:help ao/min">ao/min</a>                  -  computes the minimum value of
the data in the AO.
   <a href="matlab:help ao/minus">ao/minus</a>                -  implements minus operator for
analysis objects.
   <a href="matlab:help ao/mlpsd_m">ao/mlpsd_m</a>              -  m-file only version of the
LPSD algorithm
   <a href="matlab:help ao/mlpsd_mex">ao/mlpsd_mex</a>            -  calls the ltpda_dft.mex
to compute the DFT part of the LPSD
   <a href="matlab:help ao/mltfe">ao/mltfe</a>                -  compute log-frequency space
TF
   <a href="matlab:help ao/mode">ao/mode</a>                 -  computes the modal value of
the data in the AO.
   <a href="matlab:help ao/mpower">ao/mpower</a>               -  implements mpower operator
for analysis objects.
   <a href="matlab:help ao/mrdivide">ao/mrdivide</a>             -  implements mrdivide
operator for analysis objects.
   <a href="matlab:help ao/mtimes">ao/mtimes</a>               -  implements mtimes operator
for analysis objects.
   <a href="matlab:help ao/ngconv">ao/ngconv</a>               -  is called by the function
LTPDA_NOISEGEN
   <a href="matlab:help ao/nginit">ao/nginit</a>               -  is called by the function
LTPDA_NOISEGEN
   <a href="matlab:help ao/ngprop">ao/ngprop</a>               -  is called by the function
LTPDA_NOISEGEN
   <a href="matlab:help ao/ngsetup">ao/ngsetup</a>              -  is called by the function
LTPDA_NOISEGEN
   <a href="matlab:help ao/ngsetup_vpa">ao/ngsetup_vpa</a>          - % ALGONAME = mfilename;
   <a href="matlab:help ao/norm">ao/norm</a>                 -  overloads the norm operator
for Analysis Objects.
   <a href="matlab:help ao/phase">ao/phase</a>                -  overloads the ltpda_phase
operator for Analysis objects.
   <a href="matlab:help ao/plot">ao/plot</a>                 -  a simple plot of analysis
objects.
   <a href="matlab:help ao/plus">ao/plus</a>                 -  implements addition operator
for analysis objects.
   <a href="matlab:help ao/polyfit">ao/polyfit</a>              -  overloads polyfit()
function of MATLAB for Analysis Objects.
   <a href="matlab:help ao/power">ao/power</a>                -  implements power operator for
analysis objects.
   <a href="matlab:help ao/psd">ao/psd</a>                  -  makes power spectral density
```

```
estimates of the time-series objects
   <a href="matlab:help ao/pwelch">ao/pwelch</a>              -  makes power spectral density
estimates of the time-series objects
   <a href="matlab:help ao/rdivide">ao/rdivide</a>             -  implements rdivide operator
for analysis objects.
   <a href="matlab:help ao/real">ao/real</a>                -  overloads the real operator
for Analysis objects.
   <a href="matlab:help ao/resample">ao/resample</a>            -  overloads resample
function for AOs.
   <a href="matlab:help ao/rms">ao/rms</a>                  -  Calculate RMS deviation from
spectrum
   <a href="matlab:help ao/search">ao/search</a>              -  selects AOs that match the
given name.
   <a href="matlab:help ao/select">ao/select</a>              -  select particular samples
from the input AOs and return new AOs with only those samples.
   <a href="matlab:help ao/setDescription">ao/setDescription</a>      -  sets the
'description' property of the ao.
   <a href="matlab:help ao/setFs">ao/setFs</a>               -  sets the 'fs' property of the
ao.
   <a href="matlab:help ao/setT0">ao/setT0</a>               -  sets the 't0' property of the
ao.
   <a href="matlab:help ao/setX">ao/setX</a>                -  sets the 'x' property of the
ao.
   <a href="matlab:help ao/setXY">ao/setXY</a>               -  sets the 'xy' property of the
ao.
   <a href="matlab:help ao/setXunits">ao/setXunits</a>           -  sets the 'xunits'
property of the ao.
   <a href="matlab:help ao/setY">ao/setY</a>                -  sets the 'y' property of the
ao.
   <a href="matlab:help ao/setYunits">ao/setYunits</a>           -  sets the 'yunits'
property of the ao.
   <a href="matlab:help ao/setZ">ao/setZ</a>                -  sets the 'z' property of the
ao.
   <a href="matlab:help ao/sign">ao/sign</a>                -  overloads the sign operator
for Analysis objects.%
   <a href="matlab:help ao/sin">ao/sin</a>                  -  overloads the sin method for
Analysis objects.
   <a href="matlab:help ao/smoother">ao/smoother</a>            -  smooths a given series of
data points using the specified method.
   <a href="matlab:help ao/sort">ao/sort</a>                -  the values in the AO.
   <a href="matlab:help ao/spectrogram">ao/spectrogram</a>         -  computes a spectrogram
of the given ao/tsdata.
   <a href="matlab:help ao/spikecleaning">ao/spikecleaning</a>       -  detects and corrects
possible spikes in analysis objects
   <a href="matlab:help ao/split">ao/split</a>               -  split an analysis object into
the specified segments.
   <a href="matlab:help ao/sqrt">ao/sqrt</a>                -  computes the square root of
the data in the AO.
   <a href="matlab:help ao/std">ao/std</a>                  -  computes the standard deviation
of the data in the AO.
   <a href="matlab:help ao/string">ao/string</a>              -  writes a command string that
can be used to recreate the input Analysis object(s).
   <a href="matlab:help ao/sum">ao/sum</a>                  -  computes the sum of the data
in the AO.
   <a href="matlab:help ao/svd">ao/svd</a>                  -  overloads the determinant
function for Analysis objects.
   <a href="matlab:help ao/t0">ao/t0</a>                   -  Get the data property 't0'.
   <a href="matlab:help ao/tan">ao/tan</a>                  -  overloads the tan method for
Analysis objects.
   <a href="matlab:help ao/tfe">ao/tfe</a>                  -  makes transfer function
estimates of the time-series objects.
   <a href="matlab:help ao/timedomainfit">ao/timedomainfit</a>       -  uses lscov to fit a
set of time-series AOs to a target time-series AO.
   <a href="matlab:help ao/times">ao/times</a>               -  implements times operator for
analysis objects.
   <a href="matlab:help ao/timeshift">ao/timeshift</a>           -  for AO/tsdata objects,
shifts the time axis such that x(1) = 0.
   <a href="matlab:help ao/transpose">ao/transpose</a>           -  overloads the .' operator
for Analysis Objects.
   <a href="matlab:help ao/uminus">ao/uminus</a>              -  overloads the uminus
operator for Analysis objects.
   <a href="matlab:help ao/update_struct">ao/update_struct</a>       -  update the input
structure to the current ltpda version
   <a href="matlab:help ao/upsample">ao/upsample</a>            -  overloads upsample
function for AOs.
   <a href="matlab:help ao/validate">ao/validate</a>            -  checks that the input
Analysis Object is reproducible and valid.
   <a href="matlab:help ao/var">ao/var</a>                  -  computes the variance of the
data in the AO.
   <a href="matlab:help ao/welch">ao/welch</a>               -  Welch spectral estimation
method.
   <a href="matlab:help ao/welchparse">ao/welchparse</a>          -    Parser for the PWELCH
& SPECTROGRAM functions
   <a href="matlab:help ao/welchscale">ao/welchscale</a>          -  scales the output of
welch to be in the required units
   <a href="matlab:help ao/x">ao/x</a>                     -  Get the data property 'x'.
```

```
   <a href="matlab:help ao/xcorr">ao/xcorr</a>                - makes cross-correlation
estimates of the time-series
   <a href="matlab:help ao/xspec">ao/xspec</a>                - performs cross-spectral
analysis of various forms.
   <a href="matlab:help ao/xunits">ao/xunits</a>               - Get the data property
'xunits'.
   <a href="matlab:help ao/y">ao/y</a>                    - Get the data property 'y'.
   <a href="matlab:help ao/yunits">ao/yunits</a>               - Get the data property
'yunits'.
   <a href="matlab:help ao/zeropad">ao/zeropad</a>              - zero pads the input data
series.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ao2m(varargin)


  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/type now.');

  % Call ltpda_uoh/type
  type(varargin{:});


end

```

*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>   -  applys the given method
to the input 2D data.
   <a href="matlab:help cdata/applyoperator">cdata/applyoperator</a> -  applys 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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%% class: data2D %%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: data2D   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help data2D/applymethod">data2D/applymethod</a>   -  applys the given
method to the input 2D data.
   <a href="matlab:help data2D/applyoperator">data2D/applyoperator</a> -  applys 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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%% class: ltpda_data %%%%%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: ltpda_data   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help ltpda_data/ltpda_data">ltpda_data/ltpda_data</a> -  is the abstract
base class for ltpda data objects.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.
```

```
%%%
```

```
  %%%%
```

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setA

PURPOSE ^

SETA Set the property 'a'

SYNOPSIS ^

function obj = setA(obj, val)

DESCRIPTION ^

```
 SETA Set the property 'a'
%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setFs

PURPOSE ^

SETFS Set the property 'fs'

SYNOPSIS ^

function obj = setFs(obj, val)

DESCRIPTION ^

```
 SETFS Set the property 'fs'
%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setInfile

PURPOSE ^

SETINFILE Set the property 'infile'

SYNOPSIS ^

function obj = setInfile(obj, val)

DESCRIPTION ^

```
 SETINFILE Set the property 'infile'
%%%
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%% class: ltpda_nuo %%%%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: ltpda_nuo   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help ltpda_nuo/ltpda_nuo">ltpda_nuo/ltpda_nuo</a> -  is the abstract ltpda
base class for ltpda non user object classes.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

classdef ltpda_nuo < ltpda_obj


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Property definition                          %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %---------- Public (read/write) Properties  ----------
  properties
  end

  %---------- Protected read-only Properties ----------
  properties (SetAccess = protected)
  end

  %---------- Private Properties ----------
  properties (GetAccess = protected, SetAccess = protected)
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                            Check property setting                         %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                                Constructor                                %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  methods
    function obj = ltpda_nuo(varargin)

      %%% Call superclass
      obj = obj@ltpda_obj(varargin{:});

      %%%%%%%%%%   Set dafault values   %%%%%%%%%%
    end
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %                              Methods (public)                             %
```

```
  %%%%
```

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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'
%%%
```

setVersion

PURPOSE ^

SETVERSION Set the property 'version'

SYNOPSIS ^

function setVersion(obj, val)

DESCRIPTION ^

```
 SETVERSION Set the property 'version'
%%%
```

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% RETRIEVE retrieves a collection of objects from an LTPDA repository.
%%%
```

```
      error('### Incorrect usage');
    end
  elseif nargin > 1 && isnumeric(varargin{2})
    ids = [varargin{2:end}];
  else
    error('### Incorrect usage');
  end

  utils.helper.msg(msg.PROC1, ['retrieving objects [' num2str(ids) ']']);

  for j=1:length(ids)
    % Get object
    xdoc = utils.mysql.getXdoc(conn, ids(j));
    if ~isempty(xdoc)
      obj = utils.helper.xmlread(xdoc);
      if j==1
        objs = {obj};
      else
        objs = [objs {obj}];
      end

      % make transaction entry
      t     = time();
      tdate = t.format('yyyy-mm-dd HH:MM:SS');
      try
        message = utils.mysql.insert(conn, ...
          'transactions',...
          'obj_id', ids(j),...
          'user_id', userid,...
          'transdate', tdate,...
          'direction', 'out'...
          );
        utils.helper.msg(msg.PROC1, 'updated transactions table');
      catch
        error('### Failed to make entry in transactions table');
      end
    else
      warning('!!! Error retrieving object: %d', ids(j));
    end
  end

  % Set outputs
  if nargout == 1
    if length(objs) == 1
      varargout{1} = objs{1};
    else
      varargout{1} = objs;
    end
  else
    for k=1:nargout
      varargout{k} = objs{k};
    end
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  Local Functions                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
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, 'ltpda_uo', '', utils.const.categories.internal, '$Id: retrieve.m,v 1.4 2008/09/04 15:29:30 ingo Exp $', sets, pl);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
function plo = getDefaultPlist()
  plo = plist('conn', [], 'ids', []);
end
```

setCreated

PURPOSE ^

SETCREATED Set the property 'created'

SYNOPSIS ^

function obj = setCreated(obj, val)

DESCRIPTION ^

```
 SETCREATED Set the property 'created'
%%%
```

setCreator

PURPOSE ^

SETCREATOR Set the property 'creator'

SYNOPSIS ^

function obj = setCreator(obj, val)

DESCRIPTION ^

```
 SETCREATOR Set the property 'creator'
%%%
```

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.
%%%
```

- 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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = submit(varargin)

  %% Process inputs
  %%% Check if this is a call for parameters
  if utils.helper.isinfocall(varargin{:})
    varargout{1} = getInfo(varargin{3});
    return
  end

  import utils.const.*
  utils.helper.msg(msg.MNAME, 'running %s/%s', mfilename('class'), mfilename);

  %%% Collect all AOs
%   objs  = utils.helper.collect_objects(varargin(:), 'ltpda_uo');
  [sinfo, invars, objs] = utils.helper.collect_objects(varargin(:), 'struct');

  if isempty(objs)
    error('### Please input at least one LTPDA user object.');
  end
  if isempty(sinfo)
    error('### Please provide an input submission structure.');
```

```
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)),
obj(jj).name);
0102
0103          % Object name
0104          name = obj(jj).name;
0105
0106          % Creation time
0107          if isa(obj(jj).created, 'time')
0108            created = obj(jj).created.format('yyyy-mm-dd HH:MM:SS');
0109          else
0110            created = '1970-01-01 00:00:00';
0111          end
0112
0113          % Version
0114          objver = obj(jj).version;
0115
0116          %%%%%%%%%%% Convert object to XML
0117          % make pointer to xml document
0118          xml = com.mathworks.xml.XMLUtils.createDocument('ltpda_object');
0119          % extract parent node
0120          parent = xml.getDocumentElement;
0121          % add Attribute 'ltpda_version' to the root node
0122          ltpda_version   = getappdata(0, 'ltpda_version');
0123          parent.setAttribute('ltpda_version', ltpda_version);
0124          % write obj into xml
0125          utils.helper.xmlwrite(obj(jj), xml, parent, '');    % Save the XML document.
0126          otxt = xmlwrite(xml); % then to string
0127
0128          %%%%%%%%%%% Create MD5 hash of object
0129          md5hash = utils.prog.hash(otxt, 'MD5');
0130
0131          %%%%%%%%%%% Submit object to objs table
0132          try
0133            utils.helper.msg(msg.PROC1, 'uploading XML data...\f');
0134            curs = exec(conn, ['insert into objs(xml,hash) values(''' otxt ''', '''
char(md5hash) ''' );']);
0135            close(curs);
0136            utils.helper.msg(msg.PROC1, '\bdone.');
0137          catch
0138            utils.helper.msg(msg.PROC1, '### Server returned: %s', curs.Message);
0139            error('### Submission error');
0140          end
0141
0142          try
0143            objid = getObjID(conn, md5hash);
0144            utils.helper.msg(msg.PROC1, 'submitted object %s with id %d', class(obj(jj)),
objid);
0145          catch
```

```
          error('### Failed to get ID of submitted object.');
        end

        %%%%%%%%%%% Make objmeta entry

        % For now reference IDs are stored in a string
        if ischar(sinfo.reference_ids)
          refIds = sinfo.reference_ids;
        else
          refIds = csv(sinfo.reference_ids);
        end

        try
          message = utils.mysql.insert(conn, ...
            'objmeta',...
            'obj_id', objid,...
            'obj_type', class(obj(jj)),...
            'name', name,...
            'created', created,...
            'version', objver,...
            'ip', prov.ip,...
            'hostname', prov.hostname,...
            'os', prov.os,...
            'submitted', datestr(now, 31),...
            'experiment_title', sinfo.experiment_title,...
            'experiment_desc', sinfo.experiment_description,...
            'reference_ids', refIds,...
            'additional_comments', sinfo.additional_comments,...
            'additional_authors', sinfo.additional_authors,...
            'keywords', sinfo.keywords, ...
            'quantity', sinfo.quantity, ...
            'analysis_desc', sinfo.analysis_description...
            );

          utils.helper.msg(msg.PROC1, 'made meta-data entry');
        catch
          lasterr
          error('### failed to make meta-data entry');
        end

        %%%%%%%%%%% Update object tables
        eid = update_object_tables(conn, obj(jj), objid);

        %%%%%%%%%%% Update transactions table
        try
          message = utils.mysql.insert(conn, ...
            'transactions',...
            'obj_id', objid,...
            'user_id', userid,...
            'transdate', tdate,...
            'direction', 'in'...
            );
          utils.helper.msg(msg.PROC1, 'updated transactions table');
        catch
          error('### Failed to update transactions table');
        end

        % Collect this ID
        ids = [ids objid];
      end
    end

    %% Now make collection entry
    if ~isempty(ids)
      try
        message = utils.mysql.insert(conn, ...
          'collections',...
          'nobjs', length(ids),...
          'obj_ids', csv(ids)...
          );
        utils.helper.msg(msg.PROC1, 'made collection entry');
      catch
        error('### Failed to make an entry in the collections table');
      end
    end
  catch ME
    utils.helper.msg(msg.PROC1, '### Submission error - cleaning up.');
    rethrow(ME)
  end

  % Pass back outptus
  varargout{1} = ids;
  varargout{2} = getCollectionID(conn, ids);

  utils.helper.msg(msg.PROC1, 'submission complete.');

```

```
end

%%%
```

```
          message = utils.mysql.insert(conn, ...
            'fsdata',...
            'xunits', char(obj.xunits),...
            'yunits', char(obj.yunits));
        end
        utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
        % get the entry id
        eid = getObjID(conn, '');
      catch
        lasterr
        error('### failed to make meta-data entry for %s table', cl);
      end
      %------------------ MFIR
    case 'mfir'
      % Make the meta-data entry
      try
        if isempty(obj.fs)
          message = utils.mysql.insert(conn, ...
            'mfir',...
            'obj_id', objid,...
            'in_file', obj.infile...
            );
        else
          message = utils.mysql.insert(conn, ...
            'mfir',...
            'obj_id', objid,...
            'in_file', obj.infile,...
            'fs', obj.fs ...
            );
        end
        utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
        % get the entry id
        eid = getObjID(conn, '');
      catch
        lasterr
        error('### failed to make meta-data entry for %s table', cl);
      end
      %------------------ MIIR
    case 'miir'
      % Make the meta-data entry
      try
        if isempty(obj.fs)
          message = utils.mysql.insert(conn, ...
            'miir',...
            'obj_id', objid,...
            'in_file', obj.infile...
            );
        else
          message = utils.mysql.insert(conn, ...
            'miir',...
            'obj_id', objid,...
            'in_file', obj.infile,...
            'fs', obj.fs ...
            );
        end
        utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
        % get the entry id
        eid = getObjID(conn, '');
      catch
        lasterr
        error('### failed to make meta-data entry for %s table', cl);
      end
      %------------------ TSDATA
    case 'tsdata'
      % Make the meta-data entry
      try
        message = utils.mysql.insert(conn, ...
          'tsdata',...
          'xunits', char(obj.xunits),...
          'yunits', char(obj.yunits),...
          'fs', obj.fs,...
          'nsecs', obj.nsecs,...
          't0', format(obj.t0, 'yyyy-mm-dd HH:MM:SS') ...
          );
        utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
        % get the entry id
        eid = getObjID(conn, '');
      catch
        lasterr
        error('### failed to make meta-data entry for %s table', cl);
      end
      %------------------ XYDATA
    case 'xydata'
      % Make the meta-data entry
      try
        message = utils.mysql.insert(conn, ...
```

```
            'xydata',...
            'xunits', char(obj.xunits),...
            'yunits', char(obj.yunits) ...
            );
          utils.helper.msg(msg.PROC2, 'made meta-data entry for %s', cl);
          % get the entry id
          eid = getObjID(conn, '');
        catch
          lasterr
          error('### failed to make meta-data entry for %s table', cl);
        end
      otherwise
        utils.helper.msg(msg.PROC2, '# no special table for objects of type %s', cl);
    end

    if ~isempty(eid)
      utils.helper.msg(msg.PROC2, 'made meta-data entry in %s table with id %d', cl, eid);
    end
  end

  %%%%
```

```
  % check these
  for jj=1:length(manfields)
    if ~ismember(fnames, manfields{jj})
      for kk=1:length(manfields)
        utils.helper.msg(msg.OFF, 'Required Field: %s', manfields{kk});
      end
      error('The sinfo structure should contain at least the above fields.');
    end
  end

  % Check other fields exist
  if ~isfield(sinfo, 'quantity')
    sinfo.quantity = '';
  end
  if ~isfield(sinfo, 'keywords')
    sinfo.keywords = '';
  end
  if ~isfield(sinfo, 'reference_ids')
    sinfo.reference_ids = '';
  end
  if ~isfield(sinfo, 'additional_comments')
    sinfo.additional_comments = '';
  end
  if ~isfield(sinfo, 'additional_authors')
    sinfo.additional_authors = '';
  end

  %---- Additional checks
  if length(sinfo.experiment_title) < 5
    error('### Experiment title is mandatory and should be at least 5 characters long.');
  end
  if length(sinfo.experiment_description) < 10
    error('### Experiment description is mandatory and should be more that 10 characters long.');
  end
  if length(sinfo.analysis_description) < 10
    error('### Analysis description is mandatory and should be more that 10 characters long.');
  end

  % check all fields are present otherwise make them empty
  for jj=1:length(allfields)
    if ~ismember(fnames, allfields{jj})
      utils.helper.msg(msg.PROC2, 'setting default for field %s', allfields{jj});
      sinfo.(allfields{jj}) = '';
    end
  end

  utils.helper.msg(msg.PROC1, 'sinfo structure is valid.');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Throw description of sinfo to terminal
%
function sinfo_description
  disp('The sinfo structure should look like:');
  disp('');
  disp('  sinfo.conn                    - a database connection object');
  disp('  sinfo.experiment_title        - the experiment title       (Mandatory, >5 characters)');
  disp('  sinfo.experiment_description - the experiment description (Mandatory, >10 characters)');
  disp('  sinfo.analysis_description   - the analysis description   (Mandatory, >10 characters)');
  disp('  sinfo.quantity                - the physical quantity represented by the data');
  disp('  sinfo.keywords                - a comma-delimited list of keywords');
  disp('  sinfo.reference_ids           - IDs of other applicable LTPDA objects in the repository');
  disp('  sinfo.additional_comments     - any additional comments');
  disp('  sinfo.additional_authors      - any additional authors');
  disp('');
  error('### Incorrect specification for sinfo.')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
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, 'ltpda_uo', '', utils.const.categories.internal, '$Id:
submit.m,v 1.6 2008/09/04 15:29:30 ingo Exp $', sets, pl);
end

%%%
```

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function obj = fromDatafile(obj, pli)

  error('### It is not possible to convert this data file into a %s-object', class(obj));

end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

fromLISO

PURPOSE ^

FROMLISO Default method to read LISO files

SYNOPSIS ^

function obj = fromLISO(obj, pli)

DESCRIPTION ^

```
 FROMLISO Default method to read LISO files
%%%
```

*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
%%%
```

```
  pl = combine(pli, ii.plists);

  % Get parameters
  conn     = find(pl, 'conn');
  hostname = find(pl, 'hostname');
  database = find(pl, 'database');
  ids      = find(pl, 'id');
  cid      = find(pl, 'cid');

  % do we have a connection?
  closeConn = 0;
  if isempty(conn)
    closeConn = 1;
    % Connect to repository
    conn = utils.mysql.connect(hostname, database);
  end
  if ~isa(conn, 'database')
    error('### connection to the database failed.');
  end

  if isempty(ids) && ~isempty(cid)
    % get the ids from the cid
    ids = utils.mysql.getObjIds(conn,cid);
  end

  % Get each ID
  Nids = length(ids);
  objs = [];

  for kk=1:Nids

    %---- This id
    id = ids(kk);
    utils.helper.msg(utils.const.msg.OPROC2, 'retrieving ID %d', id);

    %---- check ID object type
    tt = utils.mysql.getObjType(conn, id);

    %---- If this is an AO
    if strcmp(tt, class(obj))
      %---- call database constructor
      obj = ltpda_uo.retrieve(conn, id);
      %---- Add history

      % make a new history object
      %  - remove connection object from plist first
      txt = textscan(conn.URL, '%s', 'Delimiter', '/', 'CollectOutput', 1);
      plc = plist('hostname', txt{1}{3}, 'database', txt{1}{4});
      pl = combine(pli, plc);
      pl.remove('conn');
      obj.addHistory(ii, pl, [], obj.hist);

      %---- Add to output array
      objs = [objs obj];
    else
      warning('    !skipping ID %d, type %s', id, tt);
    end

  end

  % close connection
  if closeConn
    close(conn);
  end
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

setHist

PURPOSE ^

SETHIST Set the property 'hist'

SYNOPSIS ^

function obj = setHist(obj, val)

DESCRIPTION ^

```
 SETHIST Set the property 'hist'
%%%
```

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%% class: plist %%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: plist   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help plist/append">plist/append</a>         -  append a param-object,
plist-object or a key/value pair to the parameter list.
   <a href="matlab:help plist/char">plist/char</a>           -  convert a parameter list into
a string.
   <a href="matlab:help plist/combine">plist/combine</a>        -  multiple parameter lists
(plist objects) into a single plist.
   <a href="matlab:help plist/copy">plist/copy</a>           -  Make copy of plist objects
depending of the second input
   <a href="matlab:help plist/display">plist/display</a>        -  display plist object.
   <a href="matlab:help plist/find">plist/find</a>           -  overloads find routine for a
parameter list.
   <a href="matlab:help plist/fromFile">plist/fromFile</a>       - Construct a ltpda_ob from
a file
   <a href="matlab:help plist/fromRepository">plist/fromRepository</a> - Construct a plist
filter from a repository
   <a href="matlab:help plist/isparam">plist/isparam</a>        -  look for a given key in
the parameter lists.
   <a href="matlab:help plist/nparams">plist/nparams</a>        -  returns the number of
param objects in the list.
   <a href="matlab:help plist/plist">plist/plist</a>          -  Plist class object
constructor.
   <a href="matlab:help plist/pset">plist/pset</a>           -  set or add a key/value pairor
a param-object into the parameter list.
   <a href="matlab:help plist/remove">plist/remove</a>         -  remove a parameter from the
parameter list.
   <a href="matlab:help plist/resp">plist/resp</a>           -  shadows miir/iirResp and
pzmodel/resp.
   <a href="matlab:help plist/string">plist/string</a>         -  converts a plist object to
a command string which will recreate the plist object.
   <a href="matlab:help plist/update_struct">plist/update_struct</a>  -  update the input
structure to the current ltpda version
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 comlpex 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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%% class: pzmodel %%%%%%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: pzmodel   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help pzmodel/abcascade">pzmodel/abcascade</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> -
%%%%%%%%%%%%%%%%%%%
```

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% class: ssm %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   class: ssm   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help ssm/assemble">ssm/assemble</a>              - assembles embedded
subsytems, 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> -
%%%%%%%%%%%%%%%%%%%
```

```
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>              -
%%%
```

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function str = char(v)
  NDimensions = numel(v.strs);
  if NDimensions == 0
    nameString = '[1]';
  else
    nameString = '';
    for nd = 1:NDimensions
      if(v.exps(nd)~=0)
        [n,d] = rat(v.exps(nd));
        if(d==1)
          if v.exps(nd) ~=1
            nameString = sprintf('%s[%s^(%g)]',nameString,v.strs{nd},v.exps(nd));
          else
            nameString = sprintf('%s[%s]',nameString,v.strs{nd});
          end
        else
          nameString = sprintf('%s[%s^(%g/%g)]',nameString,v.strs{nd},n,d);
        end
      end
    end
  end
  str = nameString;

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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);
```

```
    obj(s(1),s(2)) = unit;
    for kk=1:numel(old)
      utils.helper.msg(utils.const.msg.OPROC2, 'copying unit');
      for jj=1:length(ps)
        obj(kk).(ps{jj}) = old(kk).(ps{jj});
      end
      % reset on copy
      obj(kk).version = unit.VEROUT;
    end
  else
    obj = old;
  end
  varargout{1} = obj;
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.
%%%
```

```
  % Get unit objects
  objs = utils.helper.collect_objects(varargin(:), 'unit');

  % get display text
  txt = utils.helper.objdisp(objs);

  % display the objects
  if nargout > 0
    varargout{1} = txt;
  elseif nargout == 0;
    for j=1:numel(txt)
      disp(txt{j});
    end
  end
end

%%%
```

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 ^

```
function v = mpower(vi, exp)
  v = copy(vi, 1);
  if isa(exp, 'unit')
    exp = exp.exps;
  end
  v.exps = v.exps*exp;
  v.vals = v.vals^exp;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function v = mrdivide(v1,v2)
  v = unit;
  v.strs = [v1.strs v2.strs];
  v.exps = [v1.exps -1*v2.exps];
  v.vals = [v1.vals v2.vals];

  v = unit.simplify(v);

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function v = mtimes(v1,v2)
  v = unit;
  v.strs = [v1.strs v2.strs];
  v.exps = [v1.exps v2.exps];
  v.vals = [v1.vals v2.vals];

  v = unit.simplify(v);

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 ^

```
function v = plus(v1,v2)

  if ~strcmp(char(v1), char(v2))
    error('### Can''t add different units');
  end

  v = v1;

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function v = power(vi, exp)
  v = copy(vi,1);
  if isa(exp, 'unit')
    exp = exp.exps;
  end
  if ~isempty(exp)
    v.exps = v.exps*exp;
    v.vals = v.vals.^exp;
  end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function v = rdivide(v1,v2)
  v = unit;
  v.strs = [v1.strs v2.strs];
  v.exps = [v1.exps -1*v2.exps];
  v.vals = [v1.vals v2.vals];

  v = unit.simplify(v);

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function v = sqrt(vi)
  v = copy(vi, 1);
  v.exps = v.exps*0.5;
  v.vals = sqrt(v.vals);
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
object.
0002 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
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 $
```

```
%
% HISTORY:      29-03-2007 M Hewitson
%                  Creation
%
%%%
```

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 ^

```
function v = times(v1,v2)
  v = unit;
  v.strs = [v1.strs v2.strs];
  v.exps = [v1.exps v2.exps];
  v.vals = [v1.vals v2.vals];

  v = unit.simplify(v);

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.
%%%
```

- 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 ^

```
% MINFO a helper class for implementing units in LTPDA.
%%%
```

```
    version = '$Id: unit.m,v 1.3 2008/09/08 08:28:27 hewitson Exp $';
  end


  %----------------------------------------
  %- Public methods
  %----------------------------------------
  methods

    %----------------------------------------
    %- Constructor
    %----------------------------------------
    function u = unit(varargin)

      switch nargin
        case 0
          % Empty constructor
        case 1
          % String input

          ustr = varargin{1};

          if ischar(ustr)
            if ~isempty(ustr)

              ustr = strrep(strrep(ustr, '[', ' '), ']', ' ');

              % split on whitespace
              tks = regexp(strtrim(ustr), '\s+', 'split');
              % combine each unit
              for j=1:numel(tks)

                % Parse string
                [us, exp, val] = unit.parse(tks{j});
                u2.strs = {us};
                u2.exps = exp;
                u2.vals = val;
                u = u*u2;
              end
            end
          elseif isstruct(ustr)

            u.strs    = ustr.strs;
            u.exps    = ustr.exps;
            u.vals    = ustr.vals;
            u.version = ustr.version;
          elseif isa(ustr, 'unit')
            u = ustr;
          else
            error('### Unknown single argument constructor');
          end
      end

    end % End constructor

  end % End public methods

  %----------------------------------------
  %- Private Static methods
  %----------------------------------------
  methods (Static=true, Access=private)

    %----------------------------------------
    %- Parse a unit definition string
    %----------------------------------------
    function [us, exp, val] = parse(ustr)

      % get exp
      [s,t] = strtok(ustr, '^');
      if ~isempty(t)
        % drop any ()
        t = strrep(t(2:end),'(','');
        t = strrep(t,')','');
        exp = eval(t);
      else
        exp = 1;
      end

      % check for prefix
      if ismember(s(1), unit.supportedPrefixes) && length(s) > 1 && ismember(s(2:end), unit.supportedUnits)
        sp  = s(1);
        val = unit.prefix2val(sp);
        sm  = s(2:end);
      else
        val = 1;
```

```
        sp  = '';
        sm  = s;
      end

      % Check unit
      if ~ismember(sm, unit.supportedUnits)
        error(['### Unsupported unit: [' sp ']' sm]);
      end

      % set unit string
      us  = [sp sm];

    end % End parse

    %----------------------------------------
    %- Simplify a set of units
    %----------------------------------------
    function v = simplify(v)

      for j=1:numel(v.strs)
        for k=j+1:numel(v.strs)
          if strcmp(v.strs{j}, v.strs{k})
            v.exps(j) = v.exps(j) + v.exps(k);
            v.strs{k} = '';
            v.exps(k) = 0;
          end
        end
      end

      idx = find(v.exps~=0);
      v.strs = v.strs(idx);
      v.exps = v.exps(idx);
      v.vals = v.vals(idx);

    end

    %----------------------------------------
    %- Get the value associated with a prefix
    %----------------------------------------
    function val = prefix2val(p)
      pfxs = unit.supportedPrefixes;
      idx = find(strcmp(p, pfxs));
      val = 1e-24*10^(3*(idx-1));
    end


  end % End static private methods

  %----------------------------------------
  %- Public static methods
  %----------------------------------------
  methods (Static=true)

    function out = VEROUT()
      out = '$Id: unit.m,v 1.3 2008/09/08 08:28:27 hewitson Exp $';
    end

    function ii = getInfo(varargin)
      ii = utils.helper.generic_getInfo(varargin{:}, 'unit');
    end

    function out = SETS()
      out = {'Default'};
    end

    function out = getDefaultPlist(set)
      switch set
        case 'Default'
          out = plist();
        otherwise
          out = plist();
      end
    end


    %----------------------------------------
    %- Return a list of supported prefixes
    %----------------------------------------
    function pfxs = supportedPrefixes
      pfxs = {'y', 'z', 'a', 'f', 'p', 'n', 'u', 'm', 'c',...
        'h', 'k', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y'};
    end

    %----------------------------------------
    %- Return a list of supported units
    %----------------------------------------
```

Description of unit

```
    function pfxs = supportedUnits
      pfxs = {'m', 'kg', 's', 'A', 'K', 'mol', 'cd', ...
        'rad', 'sr', 'Hz', 'N', 'Pa', 'J', 'W', 'C', 'V', 'F', ...
        'Ohm', 'S', 'Wb', 'T', 'H', 'degC', ...
        'i', 'I', 'Count', 'A/U', 'arb'};
    end

  end
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 2008/09/05 14:10:15 hewitson Exp $

 HISTORY:     07-07-2008 Diepholz
                 Creation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = update_struct(varargin)

obj_struct = varargin{1};
struct_ver = varargin{2};

% get the version of the current toolbox
tbx_ver = strtok(getappdata(0, 'ltpda_version'));

% get only the version string without the MATLAB version
struct_ver = strtok(struct_ver);

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.0 -> 1.1'   %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
```

```
    strcmp(struct_ver, '1.0')

  try

  catch ME
    disp(varargin{1});
    throw(addCause(ME, MException('MATLAB:LTPDA','### The struct (unit) above is not from version 1.0')));
  end

  % Set the struct_ver to '1.1'
  struct_ver = '1.9.1';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.9.1' ->'1.9.2'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.9.1')

  % Set the struct_ver to '1.9.2'
  struct_ver = '1.9.2';

end

%%%%%%%%%%%%%%%%%%%%%%%%   Update version '1.2' ->'1.3'  %%%%%%%%%%%%%%%%%%%%%%%%

if utils.helper.ver2num(struct_ver) < utils.helper.ver2num(tbx_ver) && ...
    strcmp(struct_ver, '1.2')

  % Set the struct_ver to '1.3'
  struct_ver = '1.3';

end

varargout{1} = obj_struct;

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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>          - FOO
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_FOO 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>     - FOO is
the standard function template for class methods.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:                  foo(a)
0007 %              [a, b] = foo(a,pl)
0008 %
0009 % INPUTS:      a  - analysis object(s)
0010 %              pl - parameter list(s)
0011 %
```

```
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 %%%%
```

```
    sets{1} = varargin{1};
    pls = getDefaultPlist(sets{1});
  else
    sets = {'set1', 'set2', 'set3'};
    pls = [];
    for kk=1:numel(sets)
      pls = [pls getDefaultPlist(sets{kk})];
    end
  end
  % Build info object
  ii = minfo(mfilename, 'CLASS', '', 'CATEGORY', '$Id: ao_fcn_template.m,v 1.14 2008/08/07 15:37:23 ingo Exp $', sets, pls);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPlist
%
% DESCRIPTION: Get Default Plist
%
% HISTORY:     11-07-07 M Hewitson
%                Creation.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function plo = getDefaultPlist(set)
  switch set
    case 'set1'
      plo = plist();
    case 'set2'
      plo = plist('a', 1, 'b', 2);
    case 'set3'
      plo = plist('key', 'val');
    otherwise
      error('### Unknown default plist for the set [%s]', set)
  end
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_startup

% 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 $
%

%------------------------------------------------------------------------
% User Configurable parameters
%


%------------------------------------------------------------------------
% Terminal output from LTPDA
%
VERBOSE_LEVEL = 1;  % Choose the level of terminal output, higher numbers
                    % means more output; 0 is no output.

%------------------------------------------------------------------------
% Settings for SSM models
%

% Give a path or cell-array of paths
SSM_MODELS = '/Users/hewitson/working/ltp/ltp_ssm_models';

%------------------------------------------------------------------------
% Print and plot settings
%
USE_LTPDA_PRINT_SETTINGS = 'Yes';     % Choose LTPDA Print setup
USE_LTPDA_PLOT_SETTINGS  = 'Yes';     % Choose LTPDA Plot setup

%------------------------------------------------------------------------
% Various GUI settings
%
```

```
0040
0041 % 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
sets to XML
0061
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 '        \\\           /         ///      ',...
```

```
'           \\\\      //       ////          ',...
'                -------------               ',...
'                     **                     ',...
'                   ****                     ',...
};

l1 = '+-----------------------------------------------------+';
ll = length(l1);

disp(l1);
disp('|                                                     |')
for j=1:length(logo)
  disp([utils.prog.strpad(sprintf('|      %s  ', char(logo{j})), ll-1) '|']);
end
disp([utils.prog.strpad('|', ll-1) '|'])
disp([utils.prog.strpad(sprintf('|          Welcome to %s', v.Name), ll-1) '|'])
disp([utils.prog.strpad('|', ll-1) '|'])
disp([utils.prog.strpad(sprintf('|                 Version: %s', v.Version), ll-1) '|'])
disp([utils.prog.strpad(sprintf('|                 Release: %s', v.Release), ll-1) '|'])
disp([utils.prog.strpad(sprintf('|                    Date: %s', v.Date), ll-1) '|'])
disp([utils.prog.strpad('|', ll-1) '|'])
disp(l1);

%-------------------------------------------------------------------------
% set page properties for printing
if strcmpi(USE_LTPDA_PRINT_SETTINGS, 'YES')
  set(0,'DefaultFigurePaperOrientation','portrait');
  set(0,'DefaultFigurePaperType','A4');
  set(0,'DefaultFigurePaperUnits','inches');
end

%-------------------------------------------------------------------------
% Plot settings
if strcmpi(USE_LTPDA_PLOT_SETTINGS, 'YES')
  set(0,'DefaultAxesFontSize',14);
  set(0,'DefaultAxesFontWeight','bold');
  set(0,'DefaultAxesLineWidth', 1);
  set(0,'DefaultLineLineWidth', 2);
  set(0,'defaultlinemarkersize',10)
  set(0,'DefaultAxesXColor',[0.2 0.2 0.8]);
  set(0,'DefaultAxesYColor',[0.2 0.2 0.8]);
  set(0,'DefaultAxesGridLineStyle','-');
  set(0,'DefaultAxesMinorGridLineStyle','-');
  set(0,'defaultfigurenumbertitle','on');
  set(0,'DefaultFigureColor', 'w');
  set(0,'DefaultFigurePosition', [100 100 800 600]);
  set(0,'DefaultFigurePaperPositionMode','auto')

end

%-------------------------------------------------------------------------
% format of numbers on MATLAB terminal
format long g

% ----------------------------------------------------------------------
% MySQL LTPDA Repository Server Settings
%
if exist('REPOUSER', 'var') == 0
  REPOUSER = '';
end
if exist('REPOPASSWD', 'var') == 0
  REPOPASSWD = '';
end

DBDRIVER = 'com.mysql.jdbc.Driver';   % Set LTPDA Repository database driver

setappdata(0, 'ltpda_repo_user', REPOUSER);
setappdata(0, 'ltpda_repo_pass', REPOPASSWD);
setappdata(0, 'mysql_driver', DBDRIVER);
setappdata(0, 'ltpda_repo_servers', REPO_GUI_SERVERLIST);
setappdata(0, 'ltpda_gui_fontsize', GUI_FONTSIZE);
setappdata(0, 'ltpda_repo_gui_fontsize', REPO_GUI_FONTSIZE);
setappdata(0, 'ltpda_passwd_win_fontsize', PASSWD_WIN_FONTSIZE);

% add mysql JDBC driver to the java path
mysqldriver = [fileparts(which('ltpda_startup')) '/../../classes/+utils/@mysql/mysql-connector-java-5.1.6-bin.jar'];
javaaddpath(mysqldriver);

%-------------------------------------------------------------------------
% set default directories to load the data from and save the model in

setappdata(0,'DefaultDataDirectory',DEFAULT_DATA_DIR);
setappdata(0,'DefaultAnalysesDirectory',DEFAULT_ANALYSES_DIR);


```

```
try
  setdbprefs('NullStringRead','Null');
  setdbprefs('NullNumberRead', 'NaN');
catch
  warning('!!! No database toolbox is installed');
  warning('!!! You won''t be able to work with an LTPDA Repository');
end

% ---------------------------------------------------------------------
% General Variables
setappdata(0, 'wrapstringat', WRAP_STRINGS);  % wrap strings at this length for
                                               % history plot nodes
setappdata(0, 'wraplegendstringat', WRAP_LEGEND_STRINGS);  % wrap strings at this length for legends
setappdata(0, 'xmlsetsize', XML_SET_SIZE); % Max size of an xml data set <Set></Set>

setappdata(0,'ltpda_default_plot_colors', {[0.8 0.2 0.2],      ...
                                           [0.2 0.2 0.8],      ...
                                           [0.2 0.9 0.2],      ...
                                           [0.37 0.9 0.83],    ...
                                           [0.888 0.163 0.9],  ...
                                           [0 0 0],            ...
                                           [0 207 255]/255,    ...
                                           [255 128 0]/255,    ...
                                           [143 0 0]/255,      ...
                                           [255 207 0]/255,    ...
                                           [0.9 0.266 0.593]});

setappdata(0, 'timezone',         TIMEZONE);
setappdata(0, 'time_format_str', TIME_FORMAT_STR);


% ---------------------------------------------------------------------
% Version Variables

vs = ver;
for j=1:length(vs)
  v = vs(j);
  switch v.Name
    case 'MATLAB'
      matlab_version = [v.Version ' ' v.Release];
    case 'Signal Processing Toolbox'
      sigproc_version = [v.Version ' ' v.Release];
    case 'Symbolic Math Toolbox'
      ltpda_symbolic_math = [v.Version ' ' v.Release];
    case 'LTPDA Toolbox'
      ltpda_version = [v.Version ' ' v.Release];
  end
end

setappdata(0, 'matlab_version', matlab_version);
setappdata(0, 'sigproc_version', sigproc_version);
setappdata(0, 'ltpda_symbolic_math', ltpda_symbolic_math);
setappdata(0, 'ltpda_version', ltpda_version);
setappdata(0, 'ltpda_required_matlab_version', '7.6');

% -------------------------------------------------------------------
% Other settings

setappdata(0, 'ltpda_default_spectral_window', specwin('BH92', 10));
setappdata(0, 'ltpda_verbose_level', VERBOSE_LEVEL);
setappdata(0, 'ltpda_ssm_model_paths', SSM_MODELS);

if ischar(SSM_MODELS)
  SSM_MODELS = {SSM_MODELS};
end
for j=1:numel(SSM_MODELS)
  addpath(SSM_MODELS{j});
end

% ---------------------------------------------------------------------
%  import some things

import utils.const.*

% ---------------------------------------------------------------------
%  Launch some things

ltpdalauncher;


% 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 ^

```
function varargout = ltpda_foo(varargin)
% 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')
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%% ---------------- Standard history variables

ALGONAME = mfilename;
VERSION  = '$Id: ltpda_user_fcn_template.m,v 1.4 2008/03/28 11:16:46 mauro Exp $';
CATEGORY = 'Signal Processing';


%% Check if this is a call for parameters, the CVS version string
% or the function category
if nargin == 1 && ischar(varargin{1})
  in = char(varargin{1});
  if strcmp(in, 'Params')
    varargout{1} = getDefaultPL();
    return
  elseif strcmp(in, 'Version')
    varargout{1} = VERSION;
    return
  elseif strcmp(in, 'Category')
    varargout{1} = CATEGORY;
    return
  end
end


%% Collect input ao's, plist's and ao variable names
in_names = {};
for ii = 1:nargin
  in_names{end+1} = inputname(ii);
end

[as, pl, invars] = collect_inputs(varargin, in_names);

% produce one parameter list
if isa(pl, 'plist')
  pl = combine(pl, getDefaultPL());
else
  pl = getDefaultPL();
end


%% Get parameters
```

```
A = find(pl, 'A');
X = find(pl, 'X');


%% Initialize outputs
bs = [];


%% Go through analysis objects
for j=1:numel(as)
  a = as(j);

  % get data
  [x,y] = get_xy_values(a.data);

  % Do some analysis
  y = 2*A * (y + X);
  x = 1:length(x);

  %%-------- Build output AOs

  % make a new xydata object, for example
  xy = xydata(x, y);
  xy = set(xy, 'name', sprintf('ltpda_foo(%s)', a.data.name));
  xy = set(xy, 'xunits', a.data.xunits);
  xy = set(xy, 'yunits', 'Blah');

  % make a new history object
  h = history(ALGONAME, VERSION, pl, a.hist);
  h = set(h, 'invars', invars);

  % make output analysis object
  b = ao(xy, h);

  % name, mfilename, description for this object
  b = setnh(b, 'name', sprintf('ltpda_foo(%s)', invars{j}),...
    'description', find(pl,'description'));

  bs = [bs b];

end


%% Reshape the ouput to the same size of the input
varargout{1} = reshape(bs, size(as));


%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPL
%
% DESCRIPTION: Get default params
%
% HISTORY:     dd-mm-yyyy M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plo = getDefaultPL()

plo = plist('A', 10, ...
  'X', [1], ...
  'DESCRIPTION','LTPDA User Function Template');


% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpda_versions

PURPOSE ^

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
versions = {...
        '', 'SETS', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'VEROUT', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'addlistener', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'char', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'copy', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'delete', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'display', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/eq.m',
'eq', '1.8', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
'08af28fc64b0b5c7d96638545132d050', '', 'ltpda_obj'; ...
        '', 'findobj', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'findprop', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'ge', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'getDefaultPlist', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'getInfo', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'gt', '', '', '', '', '', '', '', 'ltpda_obj'; ...
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/isprop.m', 'isprop',
'1.2', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
'61cf4eab96bd5818098a39bf62a429cb', '', 'ltpda_obj'; ...
        '', 'isvalid', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'le', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'lt', '', '', '', '', '', '', '', 'ltpda_obj'; ...
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/ltpda_obj.m',
'ltpda_obj', '1.22', '2008/09/06', '12:24:27', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
'94dfb51dee1516944a68b270de6bd5aa', 'Constructor', 'ltpda_obj'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj/ne.m',
'ne', '1.5', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_obj',
'01a6311c65819cae935a156c167afffe', '', 'ltpda_obj'; ...
        '', 'notify', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'update_struct', '', '', '', '', '', '', '', 'ltpda_obj'; ...
        '', 'SETS', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'VEROUT', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'addlistener', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'char', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'copy', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'delete', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'display', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'eq', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'findobj', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'findprop', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'ge', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', 'getDefaultPlist', '', '', '', '', '', '', '', 'ltpda_nuo'; ...
        '', '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',
'a0feea29179f0a1bb40ba4c65a2a7055', '', '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', '', '', '', '', '', '', '', 'ltpda_uoh'; ...
0094          '', 'notify', '', '', '', '', '', '', '', 'ltpda_uoh'; ...
0095          '', 'retrieve', '', '', '', '', '', '', '', 'ltpda_uoh'; ...
0096          '', 'save', '', '', '', '', '', '', '', 'ltpda_uoh'; ...
0097
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh/setName.m', 'setName',
'1.4', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh',
'35067a520f154855ae47198b5993f12d', '', 'ltpda_uoh'; ...
0098          '', 'string', '', '', '', '', '', '', '', 'ltpda_uoh'; ...
0099          '', 'submit', '', '', '', '', '', '', '', 'ltpda_uoh'; ...
0100          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh/type.m',
'type', '1.5', '2008/09/04', '15:29:30', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ltpda_uoh',
'251f90e306de1c0aca4323cd3789dd03', '', 'ltpda_uoh'; ...
0101          '', 'update_struct', '', '', '', '', '', '', '', 'ltpda_uoh'; ...
0102          '', 'SETS', '', '', '', '', '', '', '', 'minfo'; ...
0103          '', 'VEROUT', '', '', '', '', '', '', '', 'minfo'; ...
0104          '', 'addlistener', '', '', '', '', '', '', '', 'minfo'; ...
0105          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo/char.m',
'char', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo',
'f87de39bfcf87bac667a07f96dfcd002', 'Output', 'minfo'; ...
0106          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo/copy.m',
'copy', '1.6', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo',
'd3e28273e2226345310788f67493dede', '', 'minfo'; ...
0107          '', 'delete', '', '', '', '', '', '', '', 'minfo'; ...
0108          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo/display.m',
'display', '1.4', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo',
'b101ad863ace7e111312e9d6e6c3d34b', 'Output', 'minfo'; ...
0109          '', 'eq', '', '', '', '', '', '', 'Relational Operator', 'minfo'; ...
0110          '', 'findobj', '', '', '', '', '', '', '', 'minfo'; ...
0111          '', 'findprop', '', '', '', '', '', '', '', 'minfo'; ...
0112          '', 'ge', '', '', '', '', '', '', '', 'minfo'; ...
0113          '', 'getDefaultPlist', '', '', '', '', '', '', '', 'minfo'; ...
0114          '', 'getInfo', '', '', '', '', '', '', '', 'minfo'; ...
0115          '', 'gt', '', '', '', '', '', '', '', 'minfo'; ...
0116          '', 'isprop', '', '', '', '', '', '', 'Helper', 'minfo'; ...
0117          '', 'isvalid', '', '', '', '', '', '', '', 'minfo'; ...
0118          '', 'le', '', '', '', '', '', '', '', 'minfo'; ...
0119          '', 'lt', '', '', '', '', '', '', '', 'minfo'; ...
0120          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo/minfo.m',
'minfo', '1.15', '2008/09/03', '16:34:49', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo',
'6699cdb4a8c770820b7609ef729b92a1', 'Constructor', 'minfo'; ...
0121          '', 'ne', '', '', '', '', '', '', 'Relational Operator', 'minfo'; ...
0122          '', 'notify', '', '', '', '', '', '', '', 'minfo'; ...
0123
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo/setMversion.m',
'setMversion', '1.3', '2008/09/04', '15:33:01', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo',
'd4f45a76dba3e08e989b4db1232bebf5', 'Internal', 'minfo'; ...
0124
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo/update_struct.m',
'update_struct', '1.2', '2008/09/05', '14:16:11', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@minfo',
'bd616f809b3876703cc80d642fd83c61', '', 'minfo'; ...
0125          '', 'SETS', '', '', '', '', '', '', '', 'unit'; ...
0126          '', 'VEROUT', '', '', '', '', '', '', '', 'unit'; ...
0127          '', 'addlistener', '', '', '', '', '', '', '', 'unit'; ...
0128          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/char.m',
'char', '', '', '', '', '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'e8056f4e00ce513fda35e18edb2229ac', '', 'unit'; ...
0129          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/copy.m',
'copy', '1.1', '2008/09/05', '14:10:15', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'c0fd7c931b0d914a6f555f302ab65f93', '', 'unit'; ...
0130          '', 'delete', '', '', '', '', '', '', '', 'unit'; ...
0131          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/display.m',
'display', '1.1', '2008/09/05', '14:10:15', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'd91a459422a0f0da7f3414bef1df0455', 'Output', 'unit'; ...
0132          '', 'eq', '', '', '', '', '', '', 'Relational Operator', 'unit'; ...
0133          '', 'findobj', '', '', '', '', '', '', '', 'unit'; ...
0134          '', 'findprop', '', '', '', '', '', '', '', 'unit'; ...
0135          '', 'ge', '', '', '', '', '', '', '', 'unit'; ...
0136          '', 'getDefaultPlist', '', '', '', '', '', '', '', 'unit'; ...
0137          '', 'getInfo', '', '', '', '', '', '', '', 'unit'; ...
0138          '', 'gt', '', '', '', '', '', '', '', 'unit'; ...
0139          '', 'isprop', '', '', '', '', '', '', 'Helper', 'unit'; ...
0140          '', 'isvalid', '', '', '', '', '', '', '', 'unit'; ...
0141          '', 'le', '', '', '', '', '', '', '', 'unit'; ...
0142          '', 'lt', '', '', '', '', '', '', '', 'unit'; ...
0143          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/mpower.m',
```

```
'mpower', '', '', '', '', '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'63b60fa6b2ef08f32653863b6b141e6d', '', 'unit'; ...
0144          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/mrdivide.m',
'mrdivide', '', '', '', '',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'bd153d7a8b7f758f5b79fc5d4a8ff973', '', 'unit'; ...
0145          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/mtimes.m',
'mtimes', '', '', '', '', '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'b2a1241ab652786760d31c9bbb5f4441', '', 'unit'; ...
0146          '', 'ne', '', '', '', '', '', '', 'Relational Operator', 'unit'; ...
0147          '', 'notify', '', '', '', '', '', '', '', 'unit'; ...
0148          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/plus.m',
'plus', '', '', '', '', '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'd5f54788a271b7167fa4be2106391753', '', 'unit'; ...
0149          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/power.m',
'power', '', '', '', '', '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'8c0c9ec250c97e6bf9aca128090c581a', '', 'unit'; ...
0150          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/rdivide.m',
'rdivide', '', '', '', '',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'3cf23d8ab411705c78425513ec216a2b', '', 'unit'; ...
0151          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/sqrt.m',
'sqrt', '', '', '', '', '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'691432674f72623651ef26365baf1139', '', 'unit'; ...
0152          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/string.m',
'string', '1.1', '2008/09/05', '14:10:15', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'539d6aecd863d4269269546095bd8cf5', 'Helper', 'unit'; ...
0153          '', 'supportedPrefixes', '', '', '', '', '', '', '', 'unit'; ...
0154          '', 'supportedUnits', '', '', '', '', '', '', '', '', 'unit'; ...
0155          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/times.m',
'times', '', '', '', '', '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'cf41d1550713239eb2c1913a2f03b3ec', '', 'unit'; ...
0156          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/unit.m',
'unit', '1.3', '2008/09/08', '08:28:27', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'664b7801213098784a921cc410e6e90c', 'Constructor', 'unit'; ...
0157
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit/update_struct.m',
'update_struct', '1.1', '2008/09/05', '14:10:15', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@unit',
'bdbf05cd3b926c86f30bedc1eb566658', '', 'unit'; ...
0158          '', 'SETS', '', '', '', '', '', '', '', 'ao'; ...
0159          '', 'VEROUT', '', '', '', '', '', '', '', 'ao'; ...
0160          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/abs.m', 'abs',
'1.33', '2008/09/06', '15:23:19', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'c5d87c4688e878bf0440b89f006dadd7', 'Operator', 'ao'; ...
0161          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/acos.m', 'acos',
'1.19', '2008/09/05', '11:05:28', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'ccb65d8f8693b92c4f2aed6585eba267', 'Trigonometry', 'ao'; ...
0162          '', 'addHistory', '', '', '', '', '', '', 'Internal', 'ao'; ...
0163          '', 'addlistener', '', '', '', '', '', '', '', 'ao'; ...
0164          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/ao.m', 'ao',
'1.158', '2008/09/07', '18:11:06', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'e04744b73298879a7fc1690f138de118', 'Constructor', 'ao'; ...
0165          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/ao2m.m', 'ao2m',
'1.26', '2008/09/08', '09:00:39', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'36bdb44e9a1cb71e97ded6d0f912963a', '', 'ao'; ...
0166          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/asin.m', 'asin',
'1.16', '2008/09/05', '11:05:28', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'0491423510d626185d778a7713119af2', 'Trigonometry', 'ao'; ...
0167          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/atan.m', 'atan',
'1.16', '2008/09/05', '11:05:28', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'51efc479d9ccbe7aeea55d4d6eec41ec', 'Trigonometry', 'ao'; ...
0168          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/atan2.m',
'atan2', '1.8', '2008/09/05', '11:05:28', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'b21f2fe9acbee1f921899514dd5990b2', 'Trigonometry', 'ao'; ...
0169          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/attachm.m',
'attachm', '1.14', '2008/09/05', '11:05:28', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'53fdd8c302fc9b0e24685b00f199f16a', 'Helper', 'ao'; ...
0170          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/attachmdl.m',
'attachmdl', '1.14', '2008/09/05', '11:05:28', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'aea6e3b3279f60c044238c8f8ac2d702', 'Helper', 'ao'; ...
0171          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/cat.m', 'cat',
'1.14', '2008/09/05', '11:05:28', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/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',
'79d2380ae5d6c02b9fad12884a37b5b6', '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/ifft.m', 'ifft',
'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/ltpda_toolbox/ltpda/classes/@ao',
'18db7133aed7a91aab34c0752b9a05f5', 'Operator', 'ao'; ...
0220          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/index.m',
'index', '1.17', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'b1eea181d83a23869daaf2aed3e18660', 'Helper', 'ao'; ...
0221          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/interp.m',
'interp', '1.24', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'c18a0677ceb1e58e1c02d2b04ebfd269', 'Signal Processing', 'ao'; ...
0222
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/interpmissing.m',
'interpmissing', '1.12', '2008/09/08', '08:31:18', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'44f2a978a215f460f0ddda05e83ab9ff', 'Signal Processing', 'ao'; ...
0223          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/inv.m', 'inv',
'1.22', '2008/09/05', '11:15:19', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'15baf0f9c677dba5646c760a7d4befb4', 'Operator', 'ao'; ...
0224          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/iplot.m',
'iplot', '1.48', '2008/09/05', '14:14:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'bfbbc281f3e05dca2296f95a82f94fac', 'Output', 'ao'; ...
0225          '', 'isprop', '', '', '', '', '', '', '', '', 'Helper', 'ao'; ...
0226          '', 'isvalid', '', '', '', '', '', '', '', '', 'ao'; ...
0227          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/join.m', 'join',
'1.24', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'6d6de67056b270ea301f90cf9df12a70', 'Helper', 'ao'; ...
0228          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/lcohere.m',
'lcohere', '1.7', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'b1a886aafe0837d07b1dc57a5f40ebc0', 'Signal Processing', 'ao'; ...
0229          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/lcpsd.m',
'lcpsd', '1.7', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'08426f3a172c2486cde8ff55e152dda8', 'Signal Processing', 'ao'; ...
0230          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/le.m', 'le',
'1.15', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'5111fefb72280ae8016532642cd4420a', 'Relational Operator', 'ao'; ...
0231          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/len.m', 'len',
'1.19', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'5483395b2033bc9fb94abf80aaa67e49', 'Helper', 'ao'; ...
0232          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/lincom.m',
'lincom', '1.9', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'a9fa4582e9fd5be15dc280162fc0a2fe', 'Signal Processing', 'ao'; ...
0233          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/linedetect.m',
'linedetect', '1.3', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'e08edf8ad4cfc3e2480ac06e3002a535', 'Signal Processing', 'ao'; ...
0234          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/ln.m', 'ln',
'1.17', '2008/09/05', '14:11:47', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'3fd5cfb97f32ad1a523f0779a3a8f00c', 'Operator', 'ao'; ...
0235          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/log.m', 'log',
'1.22', '2008/09/05', '14:11:47', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'1e98dfd8cf3ce371095ed77e1343c73e', 'Operator', 'ao'; ...
0236          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/log10.m',
'log10', '1.19', '2008/09/05', '14:11:47', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'07dc5b9c66c6ec3142b28fe34171352c', 'Operator', 'ao'; ...
0237          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/lpsd.m', 'lpsd',
'1.13', '2008/09/05', '14:14:58', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'1747c07da106ca119cc065273d1951e0', 'Signal Processing', 'ao'; ...
0238          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/lt.m', 'lt',
'1.15', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'3714c790025cbe3911ff2210ed18d4c7', 'Relational Operator', 'ao'; ...
0239          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/ltfe.m', 'ltfe',
'1.6', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'768b96e211dfcb9e59da18443ff4b20b', 'Signal Processing', 'ao'; ...
0240          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/max.m', 'max',
'1.13', '2008/09/08', '08:31:56', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao',
'63582bd38332f14cefe2afd393bfc64b', 'Operator', 'ao'; ...
0241          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ao/md5.m', 'md5',
'1.12', '2008/09/05', '11:05:29', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/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', '', '', '', '', '', '', '', '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',
'f428c65ea733f3f2420875374b69c3ae', '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',
'55d331163cddd100c67f5111b07eaf7d', '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/smoother.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',
'34e78bf5ffbfc6b697f04e895a8b82aa', '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',
'7bc0468b3c203dbee12dd988c7a3f52c', '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',
'891274777dc514214b4f3e98d4ecf1c4', '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',
'bb762c9b18aa327bff2e15b4a3d6c80b', '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/ltpda_toolbox/ltpda/classes/@plist/append.m',
'append', '1.17', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'c4c85fdcb6fe4afa04281f65683ec7bc', 'Helper', 'plist'; ...
0347          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/char.m',
'char', '1.21', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'100eccbc4ad297a1ac6c3775eeeed636', 'Output', 'plist'; ...
0348          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/combine.m',
'combine', '1.17', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'e2bb875d521c6161fbce86e7d45398d3', 'Helper', 'plist'; ...
0349          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/copy.m',
'copy', '1.7', '2008/09/03', '16:29:33', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'0a8bb40007a76e20aba4040777c209a3', '', 'plist'; ...
0350          '', 'delete', '', '', '', '', '', '', '', 'plist'; ...
0351          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/display.m',
'display', '1.11', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'09b5fd66a27fd30e7e327f7877744867', 'Output', 'plist'; ...
0352          '', 'eq', '', '', '', '', '', '', 'Relational Operator', 'plist'; ...
0353          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/find.m',
'find', '1.13', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'0cda2c3d6ead7302464bb1c13edbaaa0', 'Helper', 'plist'; ...
0354          '', 'findobj', '', '', '', '', '', '', '', 'plist'; ...
0355          '', 'findprop', '', '', '', '', '', '', '', 'plist'; ...
0356          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/fromFile.m',
'fromFile', '1.2', '2008/09/08', '09:40:46', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'87e0325e6fbc4a61a7d02fd7ae10433e', '', 'plist'; ...
0357          '', 'ge', '', '', '', '', '', '', '', 'plist'; ...
0358          '', 'getDefaultPlist', '', '', '', '', '', '', '', 'plist'; ...
0359          '', 'getInfo', '', '', '', '', '', '', '', 'plist'; ...
0360          '', 'gt', '', '', '', '', '', '', '', 'plist'; ...
0361          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/isparam.m',
'isparam', '1.9', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/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/ltpda_toolbox/ltpda/classes/@plist/nparams.m',
'nparams', '1.10', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'6cce5f90aee6eec8171c47b4697ff009', 'Helper', 'plist'; ...
0369          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/plist.m',
'plist', '1.51', '2008/09/07', '18:15:26', 'hewitson',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'c031a1b8ef50ec0366dea284029ba9f8', 'Constructor', 'plist'; ...
0370          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/pset.m',
'pset', '1.11', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'0670334d0952be20a1095a489cf256ad', 'Helper', 'plist'; ...
0371          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/remove.m',
'remove', '1.13', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'a1631d41eeb2417be59d36813aac8e20', 'Helper', 'plist'; ...
0372          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/resp.m',
'resp', '1.15', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'd107125003c05e303b6bd2ac61e18adb', 'Signal Processing', 'plist'; ...
0373          '', 'retrieve', '', '', '', '', '', '', '', 'plist'; ...
0374          '', 'save', '', '', '', '', '', '', 'Output', 'plist'; ...
0375          '', 'setName', '', '', '', '', '', '', 'Helper', 'plist'; ...
0376          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/string.m',
'string', '1.21', '2008/09/04', '15:29:31', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'ddc543d069a2569cc0efd0836bce9ab1', 'Helper', 'plist'; ...
0377          '', 'submit', '', '', '', '', '', '', 'Internal', 'plist'; ...
0378
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist/update_struct.m',
'update_struct', '1.1', '2008/07/10', '12:06:49', 'ingo',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@plist',
'd0a17bfcf7fa1f0f160996ff795c7113', '', 'plist'; ...
0379          '', 'SETS', '', '', '', '', '', '', '', 'ltpda_data'; ...
0380          '', 'VEROUT', '', '', '', '', '', '', '', 'ltpda_data'; ...
0381          '', 'addlistener', '', '', '', '', '', '', '', 'ltpda_data'; ...
0382          '', 'char', '', '', '', '', '', '', '', 'ltpda_data'; ...
0383          '', 'copy', '', '', '', '', '', '', '', 'ltpda_data'; ...
0384          '', 'delete', '', '', '', '', '', '', '', 'ltpda_data'; ...
0385          '', 'display', '', '', '', '', '', '', '', 'ltpda_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',
'd2bc399a8d770eeccd924915da71038e', '', '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', '', '', '', '', '', '', '', 'data2D'; ...
0432          '', 'SETS', '', '', '', '', '', '', '', 'cdata'; ...
0433          '', 'VEROUT', '', '', '', '', '', '', '', 'cdata'; ...
0434          '', 'addlistener', '', '', '', '', '', '', '', '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', '', 'cdata'; ...
0440          '', 'delete', '', '', '', '', '', '', '', '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', '', '', '', '', '', '', '', 'cdata'; ...
0444          '', 'findprop', '', '', '', '', '', '', '', 'cdata'; ...
0445          '', 'ge', '', '', '', '', '', '', '', 'cdata'; ...
0446          '', 'getDefaultPlist', '', '', '', '', '', '', '', 'cdata'; ...
0447          '', 'getInfo', '', '', '', '', '', '', '', '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', '', '', '', '', '', '', '', 'cdata'; ...
0450          '', 'isprop', '', '', '', '', '', '', 'Helper', 'cdata'; ...
0451          '', 'isvalid', '', '', '', '', '', '', '', 'cdata'; ...
0452          '', 'le', '', '', '', '', '', '', '', 'cdata'; ...
0453          '', 'lt', '', '', '', '', '', '', '', 'cdata'; ...
0454          '', 'ne', '', '', '', '', '', '', 'Relational Operator', 'cdata'; ...
0455          '', 'notify', '', '', '', '', '', '', '', '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',
'4a5e56a9880abc21fb58f5982d4fbeab', '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', '', 'cdata'; ...
0459          '', 'SETS', '', '', '', '', '', '', 'tsdata'; ...
0460          '', 'VEROUT', '', '', '', '', '', '', 'tsdata'; ...
0461          '', 'addlistener', '', '', '', '', '', '', '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', '', 'tsdata'; ...
0467          '', 'delete', '', '', '', '', '', '', '', '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', '', '', '', '', '', '', '', 'tsdata'; ...
0471          '', 'findprop', '', '', '', '', '', '', '', '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',
'598d05a08dbd9a3035fc9d5accc74991', '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',
'd8f1559d0bbc16161c8e9575263f0cd9', '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',
'8dfffb646f8dee12989f11e713084cd3', 'Output', 'specwin'; ...
0635          '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@specwin/copy.m',
'copy', '1.7', '2008/09/03', '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', '', '', '', '', '', '', '', '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',
'b9a47ccef9be5408270ff77c1cceede5', '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',
'22699a6d67929815f2bc33e95a4828e7', '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',
'f52e2625938d9455c5aec1d333a2244d', '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',
'e370993a10b37db37c2cdde8634f61be', '', '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', '', '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',
'5b3055ca8b91a5fa19ccefee464817ee', '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',
'e27f1a7ffb22d0d60a8a523500e400c8', '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'; ...
```

```
        '/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'; ...
'/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'; ...
        '', 'retrieve', '', '', '', '', '', '', '', 'ssm'; ...
        '', 'save', '', '', '', '', '', '', '', 'Output', 'ssm'; ...
        '', 'setName', '', '', '', '', '', '', '', 'Helper', 'ssm'; ...
        '/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm/simulate.m',
'simulate', '', '', '', '',
'/Users/hewitson/working/matlab/ltpda_toolbox/ltpda/classes/@ssm',
'acacd75d381e0d2a709ee13ca93b7ac6', 'STATESPACE', 'ssm'; ...
        '/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',
'c0a542484ddee4d782d375e550d40307', 'Constructor', 'ssm'; ...
        '/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'; ...
        '/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'; ...
        '/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'; ...
        '/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'; ...
        '', 'submit', '', '', '', '', '', '', 'Internal', 'ssm'; ...
        '', 'type', '', '', '', '', '', '', 'Output', 'ssm'; ...
'/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'; ...
        };
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:                  foo(a)
              [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 ^

```
function varargout = foo(varargin)
% FOO is the standard function template for class methods.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% DESCRIPTION: FOO is the standard function template for class methods.
%
% CALL:                   foo(a)
%              [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>
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ALGONAME = mfilename;
VERSION  = '$Id: method_fcn_template.m,v 1.6 2008/03/18 06:34:41 mauro Exp $';
CATEGORY = 'Signal Processing';


%% Check if this is a call for parameters, the CVS version string
% or the method category

% Version for standard usage
if nargin == 2 && ...
    isa(varargin{1}, 'ao') && ...
    ischar(varargin{2})
  in = char(varargin{2});
  if strcmpi(in, 'Params')
    varargout{1} = getDefaultPL();
    return
  elseif strcmpi(in, 'Version')
    varargout{1} = VERSION;
    return
  elseif strcmp(in, 'Category')
    varargout{1} = CATEGORY;
    return
  end
end


% Version for usage with sets of parameters, such as the class creators
% if (nargin == 2 || nargin == 3) && ...
%     isa(varargin{1}, 'ao') && ...
%     ischar(varargin{2})
%   in = char(varargin{2});
%   if strcmp(in, 'Params')
%     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
0113 %% Get parameters
0114 A = find(pl, 'A');
0115 X = find(pl, 'X');
0116
0117
0118 %% Initialize outputs
0119 bs = [];
0120
0121
0122 %% Go through analysis objects
0123 for j=1:numel(as)
0124   a = as(j);
0125
0126   % get data
0127   [x,y] = get_xy_values(a.data);
0128
0129   % Do some analysis
0130   y = 2*A * (y + X);
0131   x = 1:length(x);
0132
0133   %%-------- Build output AOs
0134
0135   % make a new xydata object, for example
0136   xy = xydata(x, y);
0137   xy = set(xy, 'name', sprintf('foo(%s)', a.data.name));
0138   xy = set(xy, 'xunits', a.data.xunits);
0139   xy = set(xy, 'yunits', 'Blah');
0140
0141   % make a new history object
0142   h = history(ALGONAME, VERSION, pl, a.hist);
0143   h = set(h, 'invars', invars);
0144
0145   % make output analysis object
0146   b = ao(xy, h);
0147
0148   % name, mfilename, description for this object
0149   b = setnh(b, 'name', sprintf('foo(%s)', invars{j}),...
0150     'description', find(pl,'description'));
0151
0152   bs = [bs b];
0153
0154 end
0155
0156
0157 %% Reshape the ouput to the same size of the input
0158 varargout{1} = reshape(bs, size(as));
0159
0160
0161 %% %%%%
```

```
%                   Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plo = getDefaultPL()

plo = plist('A', 10, ...
  'X', [1], ...
  'DESCRIPTION','LTPDA Method Template');




%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% FUNCTION:    getDefaultPL
%
% DESCRIPTION: Get default params, version for usage with sets
%
% HISTORY:     dd-mm-yyyy M Hewitson
%                 Creation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function out = getDefaultPL(varargin)
%
% % list of available parameter sets
% sets = {'From XML file', 'From ASCII file', 'Function', 'Values',...
%         'Time-series Function', 'From Window', 'Waveform'};
%
% if nargin == 0
%   out = sets;
%   return
% end
%
% set = varargin{1};
%
% switch set
%   case 'From XML file'
%     out = plist('filename', 'foo.xml');
%   case 'From ASCII file'
%     out = plist('filename', 'foo.txt',...
%                 'type', 'tsdata',...
%                 'columns', [1 2],...
%                 'num_columns', 10,...
%                 'comment_char', '%');
%   case 'Function'
%     out = plist('fcn', 'randn(100,1)');
%   case 'Values'
%     out = plist('vals', [1 2 3]);
%   case 'Time-series Function'
%     out = plist('tsfcn', 'randn(size(t))', 'fs', 10, 'nsecs', 100);
%   case 'From Window'
%     out = plist('win', specwin('Hanning', 100));
%   case 'Waveform'
%     out = plist('waveform', 'sine wave',...
%                 'f', 1.23,...
%                 'phi', 0,...
%                 'fs', 10,...
%                 'nsecs', 10);
%   otherwise
%     out = plist();
% end

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
oveloaded 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
oveloaded 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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function h=ancestor(obj, type)
% ANCESTOR function overloaded as JCONTROL method
%
% Example:
% h=ancestor(obj, type)
%
% returns the appropriate ancestor of the obj.hgcontainer
%
% See also builtin/ancestor
%

h=ancestor(obj.hgcontainer, type);
return
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 ^

```
function close(obj)
% 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-
% ---------------------------------------------------------------------------

delete(obj)
return
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function delete(obj)
% 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
% --------------------------------------------------------------------------

delete(obj.hgcontainer);
return
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function s=get(obj, property)
% 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
% -----------------------------------------------------------------------

% No property: list contents
if nargin==1
    if numel(obj)==1
        s.hgcontainer=obj.hgcontainer;
        s.hgcontrol=obj.hgcontrol;
        s.hghandle=obj.hghandle;
    else
        display(obj);
    end
    return
end
```

```

% Must specify a single property
if nargin>2
    error('Too many input arguments');
end

% Ensure lower-case
property=lower(property);

% Vector of JCONTROLs on input
% Call GET on each in turn returning a cell array
if numel(obj)>1
    if nargout>0
        % Pre-allocate
        s=cell(1,numel(obj));
        for idx=1:numel(obj)
            % Recursive call
            s{idx}=get(obj(idx),property);
        end
        return
    else
        % Stop problems in nested gets e.g. (get(get,...)...)
        error('Must specify a left-hand side assignment with vector input');
    end
end

% Check for ambiguity in property name
if isprop(obj.hgcontainer, property) &&...
        isprop(obj.hgcontrol, property)
    if strcmp(property,'visible')
        % Visible is an exception -  take this from the container, MATLAB
        % links this property for the container and object
        s=obj.hgcontainer.Visible;
        return
    else
    error('Shared property name ''%s''\nYou must explicitly specify the target object',...
        property);
    end
end

% Do the work
switch property
    case 'hghandle'
        s=obj.hghandle;
    case 'hgcontainer'
        s=obj.hgcontainer;
    case 'hgcontrol'
        s=obj.hgcontrol;
    otherwise
        if isprop(obj.hgcontainer, property)
            % container property
            s=obj.hgcontainer.(property);
        elseif isprop(obj.hgcontrol, property)
            % control property
            s=obj.hgcontrol.(property);
        elseif isempty(obj.hghandle)
            % Empty -default object
            s=[];
        else
            % Error
            error('No such property');
        end
end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function data=getappdata(obj, name)
% 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
% --------------------------------------------------------------------------

if nargin==1
    data=getappdata(obj.hgcontrol);
else
    data=getappdata(obj.hgcontrol, name);
end
return
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

isappdata

PURPOSE ^

ISAPPDATA function oveloaded for the JCONTROL class

SYNOPSIS ^

function isappdata(obj, name)

DESCRIPTION ^

```
ISAPPDATA function oveloaded for the JCONTROL class

Examples:
isappdata(obj)
isappdata(obj, FieldName);

See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/GETAPPDATA
JCONTROL/ISAPPDATA ISAPPDATA

---------------------------------------------------------------------------
Author: Malcolm Lidierth 07/07
Copyright  The Author & King's College London 2007
---------------------------------------------------------------------------
```

CROSS-REFERENCE INFORMATION ^

This function calls:

- isappdata ISAPPDATA function oveloaded for the JCONTROL class

This function is called by:

- isappdata ISAPPDATA function oveloaded for the JCONTROL class

SOURCE CODE ^

```
function isappdata(obj, name)
% ISAPPDATA function oveloaded for the JCONTROL class
%
% Examples:
% isappdata(obj)
% isappdata(obj, FieldName);
%
% See also: JCONTROL, JCONTROL/GETAPPDATA, JCONTROL/GETAPPDATA
% JCONTROL/ISAPPDATA ISAPPDATA
%
% ---------------------------------------------------------------------------
% Author: Malcolm Lidierth 07/07
% Copyright  The Author & King's College London 2007
% ---------------------------------------------------------------------------

isappdata(obj.hgcontrol, name);
return
end
```

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, ProeprtyValue2....);
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
PropertName/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
 <a href="http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?
objectId=14583&objectType=file">uicomponent</a> 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-contructed 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 % PropertName/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',...
```

```
%                       '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
% <a href="http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14583&objectType=file">uicomponent</a> 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-contructed java objects on input

if nargin==0
    % Default constructor
    obj.hgcontrol=[];
    obj.hghandle=[];
    obj.hgcontainer=[];
    obj=class(orderfields(obj),'jcontrol');
    return
end


% Check parent
if ishandle(Parent) && Parent==0
    % Root given as Parent-create a new figure
    container=figure('MenuBar','none');
elseif strcmp(class(Parent),'jcontrol');
    % Parent is a jcontrol - use the hgcontainer
    container=Parent.hgcontainer;
else
    % Use as specified
    container=Parent;
end

% Check the Parent is a valid handle
if ishandle(container)==0
    error('Parent is not a valid handle (has the parent figure been closed?)');
end

% Java object
if ischar(Style)
    % Create a default object
    javaobject=javaObject(Style);
elseif isjava(Style)
    % or use supplied object
    javaobject=Style;
end

% Check we have a valid Style
if isa(javaobject,'java.awt.Window')
    error('%s is a top-level container: \n%s\n',...
        Style, 'it can not have a MATLAB figure as parent/ancestor');
end

% If so, add callbacks and place in container
[obj.hgcontrol containerHandle]=javacomponent(javaobject,[],container);
% Put a copy of the container handle in obj....
obj.hghandle=containerHandle;
% ... and in the control
s=schema.prop(obj.hgcontrol ,'hghandle','mxArray');
obj.hgcontrol.hghandle=containerHandle;
s.AccessFlags.PublicSet='off';

```

```
% Put the container in obj
obj.hgcontainer = handle(containerHandle);

% Construct the instance
obj=class(orderfields(obj),'jcontrol');

% Put a reference to the hgcontrol in the UserData area of the handle
set(containerHandle, 'UserData', obj.hgcontrol);

% Set the properties and return
% Default to normalized
set(obj, 'Units','normalized');
% Set values as requested
if ~isempty(varargin)
    set(obj,varargin{:});
    return
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 ^

```
function rmappdata(obj, name)
% 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
% ----------------------------------------------------------------------------

rmappdata(obj.hgcontrol, name);
return
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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/valuename 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 Lidierth 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 ^

```
function varargout=set(varargin)
% SET method overloaded for JCONTROL class
%
% Examples:
% set(obj,PropertyName, PropertyValue)
% set(obj, PropertyName1, Value1, PropertyName2, Value2....)
%
% The propertyname/valuename 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 Lidierth 07/07
% Copyright  The Author & King's College London 2007
% ---------------------------------------------------------------------------

% Note: The usual assignin(...) call is not needed here because both the
% hgcontainer and the hgcontrol are passed by reference rather than by
% value. Changes here affect the original JCONTROL contents, and all copies
% of them in all MATLAB workspaces.

obj=varargin{1};

if nargin==1
    % Return setable properties
    s=set(obj.hgcontainer);
    s.hgcontrol=set(obj.hgcontrol);
    varargout{1}=s;
    return
end

% Otherwise make sure we have enough inputs
if nargin==2 && isstruct(varargin{2})
    % Property/value pairs in structure
    % Get the names
    propnames=fieldnames(varargin{2});
    % get the values
    propvalues=cell(length(propnames),1);
    for i=1:length(propnames)
        propvalues{i}=varargin{2}.(propnames{i});
    end
    % Recursive call to set
    set(varargin{1}, propnames, propvalues);
    return
elseif nargin==3 && iscell(varargin{2}) && iscell(varargin{3})
    % set(obj, pn, pm) where pn and pm are property and value cell vectors
    if numel(varargin{2})~=numel(varargin{3})
        error('Multiple values for each property not supported'); %#ok<ERTAG>
    end
    proplist=cell(1,2*length(varargin{2}));
    count=1;
    for i=1:length(varargin{2})
        proplist(count)=varargin{2}(i);
        proplist(count+1)=varargin{3}(i);
        count=count+2;
    end
else
    % Allow cell arrays in the property/values pairs list
    % Expand these where present
    pcount=1;
    vcount=2;
    proplist={};
    for i=1:(length(varargin)-1)/2
        if ischar(varargin{vcount})
            % Property/Value pair
            proplist{pcount}=varargin{vcount};
            proplist{pcount+1}=varargin{vcount+1};
            pcount=pcount+2;
            vcount=vcount+2;
        elseif iscell(varargin{vcount})
            % Cell array
            proplist=[proplist varargin{vcount}]; %#ok<AGROW>
            pcount=length(proplist)+1;
            vcount=vcount+1;
        end
    end
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                         proplist{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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 Lidierth 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 ^

```
function setappdata(obj, name, data)
% 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 Lidierth 07/07
% Copyright  The Author & King's College London 2007
% ---------------------------------------------------------------------------

% Revisions:
% 17.08.07  Check class of obj. This method gets called when data is a
%           jcontrol objectrather than obj

if strcmpi(class(obj),'jcontrol')
    setappdata(obj.hgcontrol, name, data);
else
    % Get here because data is a jcontrol object
    builtin('setappdata', obj, name, data)
```

```
end
return
end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function obj=subsasgn(obj, index, val)
% 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
% ----------------------------------------------------------------------------

% Note: Left-hand assignments are not needed with dot assignments
% because both the hgcontainer and the hgcontrol are passed by reference
% rather than by value.
% With () calls, left-hand assignments are needed when initializing an
% element. In these cases the caller workspace will contain a reference to
```

```
% the JCONTROL set up by subsasgn or passed to subsasgn by MATLAB in val.
% In all cases, changes here to JCONTROL properties affect the original
% JCONTROL contents, and all copies of them in all MATLAB workspaces.

switch index(1).type
    case '.'
        switch lower(index(1).subs)
            case {'hgcontainer' 'hgcontrol' 'hghandle'}
                if length(index)==1 || strcmp(index(1).subs,'hghandle')
                    error('The %s property is not settable',index(1).subs);
                else
                    subsasgn(obj.(index(1).subs), index(2:end), val);
                end
            otherwise
                % obj.property/method where the property could be in hgcontainer
                % or hgcontrol. Find out which and invoke appropriate subsasgn
                if isprop(obj.hgcontainer, index(1).subs) &&...
                        isprop(obj.hgcontrol, index(1).subs)
                    if strcmpi(index(end).subs,'visible')
                        % Set Visible property in the container, MATLAB will
                        % update the hgcontrol
                        obj.hgcontainer.Visible=VisibleProperty(val);
                        return
                    else
                    error('Shared property name ''%s''\nYou must explicitly specify the target object',...
                        index(1).subs);
                    end
                end
                if isprop(obj.hgcontainer, index(1).subs)
                    % hgcontainer property
                    subsasgn(obj.hgcontainer, index, val);
                elseif isprop(obj.hgcontrol, index(1).subs) || ismethod(obj.hgcontrol,index(1).subs)
                    % hgcontrol property or method
                    subsasgn(obj.hgcontrol, index, val);
                else
                    error('No such property or method');
                end
        end
    case '()'
        % obj is empty so initialize
        if isempty(obj)
            % Assign first element - avoids conversion to double errors
            % when initializing with subscripts
            obj=jcontrol();
            % Set the target element - need not already exist
            obj(index(1).subs{1})=val;
            return
        end

        if strcmp(class(val),'jcontrol')
            % Initializing or adding element to an array
            obj(index(1).subs{1})=val;
        else
            % Assigning field to pre-existing element
            if isvector(obj) && length(index(1).subs)==1 &&...
                    index(1).subs{1}<=length(obj)
                subsasgn(obj(index(1).subs{1}), index(2:end), val);
            elseif ~isvector(obj) || length(index(1).subs)>1
                error('Scalar or vector JCONTROL and index required on input');
            elseif index(1).subs{1}>length(obj)
                error('Index exceeds length of JCONTROL vector');
            end
        end
end
return
end
```

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 ^

```
function ret=subsref(obj, index)
% 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
% -----------------------------------------------------------------------

% Revisions:
% 21.08.07    Methods now called properly when index(1).subs=='hgcontrol'.
%             Previously subsref(obj.hgcontrol,...) was called from here with
%             index(1:end) instead of index(2:end).
% 21.08.07    Tests nargout before calling hgcontrol methods. No output is
%             requested if nargout==0. This prevents "One or more output
%             arguments not assigned..." errors being generated internally
%             e.g calling setXXXX calls. These errors will still be generated
%             (as they should)if an output is requested when none is
%             available. See 09.09.07 below.
% 09.09.07    Improve above fix with try/catch blocks. Now getXXXX calls
%             return in ans if nargout==0
% 20.09.07    Nest try/catch blocks to reduce isprop() and ismethod()
%             calls. This is substantially faster.

switch index(1).type
    case '.'
        switch lower(index(1).subs)
            case 'hgcontainer'
                if length(index)==1
                    % obj.hgcontainer
                    ret=obj.hgcontainer;
                elseif isprop(obj.hgcontainer, index(2).subs)
                    % obj.hgcontainer.property
                    ret=subsref(obj.hgcontainer,index(2:end));
                elseif isempty(obj.hgcontainer)
                    % Empty default object?
                    ret=[];
                else
                    % Otherwise no property with this name
                    error('No appropriate property %s',index(2).subs);
                end
            case 'hgcontrol'
                if length(index)==1
                    % obj.hgcontrol
                    ret=obj.hgcontrol;
                else
                    try
                        ret=subsref(obj.hgcontrol, index(2:end));
                    catch
                        try
                            CheckErr();
                            subsref(obj.hgcontrol, index(2:end));
                        catch
                            if isempty(obj.hghandle)
                                % Maybe subsref failed because obj has
                                % empty properties
                                ret=[];
                            elseif ~isprop(obj.hghandle, index(2).subs) &&...
                                    ~ismethod(obj.hghandle, index(2).subs)
                                error('jcontrol:subsref: No such property or method');
                            else
                                error('jcontrol:subsref: Unexpected error');
                            end
                        end
                    end
                end
            case 'hghandle'
                ret=obj.hghandle;
            otherwise
                % obj.property/method where the property could be in hgcontainer
                % or hgcontrol. Find out which and invoke subsref
                % recursively
                if isprop(obj.hgcontainer, index(1).subs) &&...
                        isprop(obj.hgcontrol, index(1).subs)
                    % Visible is an exception -  take this from the container, MATLAB
                    % links this property for the container and object
                    if strcmpi(index(1).subs,'visible')
                        ret=obj.hgcontainer.Visible;
                        return
                    else
                    error('Shared property name ''%s''\nYou must explicitly specify the
target object',...
                        index(1).subs);
                    end
                end

                if isprop(obj.hgcontainer, index(1).subs)
                    % hgcontainer property
                    ret=subsref(obj.hgcontainer, index);
                else
                    % hgcontrol property or method?
```

```
                    try
                        ret=subsref(obj.hgcontrol, index);
                    catch
                        try
                            CheckErr();
                            subsref(obj.hgcontrol, index);
                        catch
                            if isempty(obj.hghandle)
                                % Maybe subsref failed because obj has
                                % empty properties
                                ret=[];
                            elseif ~isprop(obj.hghandle, index(1).subs) &&...
                                    ~ismethod(obj.hghandle, index(1).subs)
                                error('jcontrol:subsref: No such property or method');
                            else
                                error('jcontrol:subsref: Unexpected error');
                            end
                        end
                    end
                end
        end
    case '()'
        % array of jcontrols
        if length(index)==1
            % One or more elements or a JCONTROL array/matrix wanted
            ret=obj(index(1).subs{:});
        else
            obj=obj(index(1).subs{:});
            if numel(obj)==1
                ret=subsref(obj, index(2:end));
            else
                error('Single element of ''%s'' must be specified', inputname(1));
            end
        end
end
return
end

%-------------------------------------------------------------------------
function CheckErr()
% CheckErr checks that the exception has been thrown for the expected
% reason - if not rethrow it.
err=lasterror();
if strcmp(err.identifier,'MATLAB:unassignedOutputs');
    return
else
    rethrow(err);
end
return
end
%-------------------------------------------------------------------------
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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</a> -  -
helper function for JCONTROL methods
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function val=VisibleProperty(val)
% 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
% ---------------------------------------------------------------------------

switch val
    case 0
        val='off';
    case 1
        val='on';
    %otherwise
        % return unchanged
end
return
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 througth 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] = getcNodevalue(cNode, s)
- function c_str = split_by_comma(str, pref)

SOURCE CODE ^

```
function ltpda_explorer(varargin)
% LTPDA_EXPLORER for exploring analysis objects and plotting/displaying their fields
%
%%%
```

```
INFO_dX = PLOT_dX / N_INFOS;
INFO_dY = .034;

BG_COLOR = [.925 .914 .847];

MAX_HIST_DEPTH = 4;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            Check the input            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Get AO list from workspace
vars = evalin('base','who');
aoList = [];
for j=1:length(vars)
  var = evalin('base', vars{j});
  if isa(var, 'ao')
    % add to list
%     aoList = [aoList var];
  end
end

obj       = [];
obj_value = {};
obj_name  = {};

% Read the ao's from the 'base' workspace

if nargin == 0
  tree_name = 'obj_ws';
  ws_vars   = evalin('base','whos');

elseif nargin == 1
  tree_name = inputname(1);
  tmp       = varargin{1};

  if iscell(tmp)
    ws_vars = [];
    for jj = 1:length(tmp)
      ws_vars(jj).name  = [inputname(1) '{' num2str(jj) '}'];
      ws_vars(jj).class = class(tmp{jj});
      ws_vars(jj).obj   = tmp{jj};
    end

  else
    ws_vars      = whos('tmp');
    ws_vars.name = inputname(1);
    ws_vars.obj  = tmp;
  end

else
  error ('##########');
end

for ii=1:length(ws_vars)

  if nargin == 0
    obj = evalin('base', ws_vars(ii).name);
  elseif nargin == 1
    obj = ws_vars(ii).obj;
  else
    error('#####');
  end

  if utils.helper.isobject(obj)

    % the object in the workspace is a single value
    if numel(obj) == 1
      obj_value{end+1} = obj;
      obj_name{end+1}  = [ws_vars(ii).class ':' ws_vars(ii).name];
    else

      [n,m] = size(obj);

      % the ao in the workspace is a vector
      if n == 1 || m == 1

        for jj=1:length(obj)
          obj_value{end+1} = obj(jj);
          obj_name{end+1}  = [ws_vars(ii).class ':' ws_vars(ii).name '(' num2str(jj) ')'];
        end

        % the ao in the workspace is a matrix
      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)
',' num2str(hh) ')'];
0168            end
0169          end
0170
0171        else
0172          error ('### this should not happen.');
0173        end
0174
0175      end
0176
0177    end
0178  end
0179
0180
0181  fig_name = ['explore the object: ' tree_name];
0182
0183  % 'Units',        'normalized',      ...
0184
0185  % Define figure
0186  fig = figure('NextPlot',     'add',             ...
0187               'NumberTitle', 'off',             ...
0188               'Toolbar',     'none',            ...
0189               'name',         fig_name,         ...
0190               'Color',        BG_COLOR,         ...
0191               'ToolBar',      'none',            ...
0192               'NextPlot',     'new',             ...
0193               'MenuBar',      'none',...
0194               'Position',    [FIG_X  FIG_Y      ...
0195                               FIG_dX FIG_dY]);
0196
0197  %%%%
```

```
                  'Position',         [(INFO_X+2*INFO_dX) INFO_Y  ...
                                        INFO_dX           INFO_dY],...
                  'BackgroundColor',   BG_COLOR);
% Define the info fields 'value'
txt4 = uicontrol('String',          '',                                ...
                  'Units',          'normalized',                      ...
                  'Style',          'Edit',                            ...
                  'Position',         [  INFO_X  (INFO_Y-INFO_dY) ...
                                       3*INFO_dX  INFO_dY],        ...
                  'BackgroundColor',   BG_COLOR);

% Define the info fields 'value'
txt5 = uicontrol('String',          '',                      ...
                  'Units',          'normalized',            ...
                  'Style',          'listbox',               ...
                  'Visible',        'off',                   ...
                  'Fontsize',        8,                      ...
                  'Position',       [DISP_X   DISP_Y         ...
                                     DISP_dX  DISP_dY],      ...
                  'BackgroundColor',  BG_COLOR);



% Define the decription of the info fields
col1 = uicontrol('String',          'Name',                          ...
                  'Units',          'normalized',                    ...
                  'Style',          'Text',                          ...
                  'Position',         [INFO_X  (INFO_Y+INFO_dY)...
                                       INFO_dX  INFO_dY],       ...
                  'BackgroundColor',  BG_COLOR);
col2 = uicontrol('String',          'Size',                                ...
                  'Units',          'normalized',                          ...
                  'Style',          'Text',                                ...
                  'Position',         [(INFO_X+INFO_dX) (INFO_Y+INFO_dY)...
                                        INFO_dX           INFO_dY],       ...
                  'BackgroundColor',   BG_COLOR);
col3 = uicontrol('String',          'Class',                                  ...
                  'Units',          'normalized',                             ...
                  'Style',          'Text',                                   ...
                  'Position',         [(INFO_X+2*INFO_dX) (INFO_Y+INFO_dY)...
                                        INFO_dX            INFO_dY],       ...
                  'BackgroundColor',   BG_COLOR);

% Define the name of the explorer
expl_name = uicontrol('String',          'Analysis Object explorer',  ...
                      'Units',           'normalized',                ...
                      'Style',           'text',                      ...
                      'Position',         [EXPL_NAME_X  EXPL_NAME_Y   ...
                                           EXPL_NAME_dX EXPL_NAME_dY],...
                      'ForeGroundColor', [0.2 0.4 1],                 ...
                      'FontSize',         18,                         ...
                      'FontWeight',      'bold',                      ...
                      'FontAngle',       'italic');

tree_menu  = uicontextmenu();
tree_menu1 = uimenu(tree_menu, 'Label',   'Plot', ...
                               'Callback', @f_tree_menu1);
tree_menu2 = uimenu(tree_menu, 'Label',   'Display', ...
                               'Callback', @f_tree_menu2);

disp_menu  = uicontextmenu;
disp_menu1 = uimenu(disp_menu, 'Label',   'close', ...
                               'Callback', @f_disp_menu1);

set(tree.Tree, 'MousePressedCallback', @mouse_cb);
set(tree.Tree, 'UIContextMenu', tree_menu);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mouse Pressed Handler
    function mouse_cb(h, ev)
        if ev.getModifiers()== ev.META_MASK
          % Workaround to set the y position
          % Workaround to set the x position
          vis = get(get(ev, 'Component'), 'VisibleRect');
          x_width  = get(get(ev, 'Component'), 'Width');
          y_height = get(get(ev, 'Component'), 'Height');

          new_x =  ev.getX-vis(1);
          new_y = -ev.getY+y_height-(y_height-vis(4));
          set(tree_menu, 'Position',  [new_x new_y], ...
                         'Visible',  'on');
        end

    end
%%%%%%%%%%%%%%%%%%%
```

```
      set(txt5, 'Visible', 'off')
      set(tree, 'position', [TREE_X  TREE_Y    ...
                             TREE_dX TREE_dY])
    end
%%%
```

```
0420
0421        end
0422
0423      end
0424 %%%%
```

```
            elseif isa(val, 'tsdata')
              str = 'Time-Series Object';
            elseif isa(val, 'xydata')
              str = 'X-Y Data Object';

            elseif isa(val, 'history')
              str = 'History Object';
              if length(val) == 1
                pl = plist(param('max_depth', MAX_HIST_DEPTH));
                plot(haxes, val, pl);
                ii = strfind(plotted, 'inhists');
                title(haxes, sprintf('History-Level: %d', length(ii)+1))
              else
                c_str{1} = 'Select the left or right branch.';
              end

            elseif isa(val, 'param')
              if length(val) == 1
                str   = char(val);
                c_str = split_by_comma(str);
              else
                for ii = 1:length(val)
                  str = char(val(ii));
                  c_str1 = split_by_comma(str, '- ');
                  c_str(end+1:end+length(c_str1)) = c_str1;
                end
              end
              str = 'Parameter Object';

            elseif isa(val, 'plist')
              if numel(val) > 1
                c_str{1} = 'select a plist';
              else
                for ii=1:length(val.params)

                  str = char(val.params(ii));
                  c_str1 = split_by_comma(str, '- ');
                  c_str(end+1:end+length(c_str1)) = strrep(c_str1, '_', '\_');
                end
              end
              str = 'Parameter List Object';

            elseif isa(val, 'mfir')
              str = 'FIR Filter Object';
            elseif isa(val, 'miir')
              str = 'IIR Filter Object';
            elseif isa(val, 'pole')
              str = 'Pole Object';
            elseif isa(val, 'provenance')
              str = 'Provenance Object';
            elseif isa(val, 'pzmodel')
              str = 'Pole Zero Object';
            elseif isa(val, 'specwin')
              str = 'Spectral Window Object';
            elseif isa(val, 'zero')
              str = 'Zero Object';
            elseif isa(val, 'time')
              str = 'Time Object';
            else
              str = 'Unknown class';
            end

          elseif iscell(val)
            for i = 1:min(length(val),3)
              if ischar(val{i})
                str = strcat(str, val{i});
              elseif isnumeric(val)
                str = strcat(str, num2str(val{i}));
              end
              if i < min(length(val),3)
                str = strcat(str, ',');
              end
            end
            if length(val) > 3
              str = strcat(str,'...');
            end

          end

          if ~isempty(c_str)
            c_str = strtrim(c_str);
            na = text(0.5,0.5,c_str);
            txt_extent = get(na, 'Extent');
            set(na, 'Position',            [0.5-txt_extent(3)/2, 0.5], ...
                    'HorizontalAlignment', 'left', ...
                    'Color',               'k',    ...
```

```
                    'FontWeight',           'bold', ...
                    'EdgeColor',            'k',    ...
                    'BackgroundColor',      'w',    ...
                    'Fontsize',             10,     ...
                    'Margin',               5);
          end

        else % ~isempty(val)
          str = 'The field is empty';
        end
        set(txt4, 'string', str)
        set(tmp, 'UserData', cell_Data);
      end


%%%
```

```
        fnames = fieldnames(val);

        for i=1:length(fnames)
          count = count + 1;
          x = getfield(val,fnames{i});

          if isa(x, 'ao')
            iconpath =[pth,'analysis_object.gif'];
          elseif isa(x, 'tsdata')
            iconpath =[pth,'ts_data.gif'];
          elseif isa(x, 'fsdata')
            iconpath =[pth,'fs_data.gif'];
          elseif isa(x, 'xydata')
            iconpath =[pth,'xy_data.gif'];
          elseif isa(x, 'cdata')
            iconpath =[pth,'c_data.gif'];
          elseif isa(x, 'history')
            iconpath =[pth,'history.gif'];
          elseif isa(x, 'plist')
            iconpath =[pth,'plist.gif'];

          elseif isstruct(x)
            if length(x) > 1
              iconpath =[pth,'structarray_icon.gif'];
            else
              iconpath =[pth,'struct_icon.gif'];
            end
          elseif isnumeric(x)
            iconpath =[pth,'double_icon.gif'];
          elseif iscell(x)
            iconpath =[pth,'cell_icon.gif'];
          elseif ischar(x)
            iconpath =[pth,'char_icon.gif'];
          elseif islogical(x)
            iconpath =[pth,'logic_icon.gif'];
          elseif isobject(x)
            iconpath =[pth,'obj_icon.gif'];
          else
            iconpath =[pth,'unknown_icon.gif'];
          end

          if isstruct(x) || isobject(x)
            isLeaf = 0;
          else
            isLeaf = 1;
          end

          nodes(count) = ltpda_uitreenode(fnames{i}, fnames{i}, iconpath, isLeaf);
        end
      end

      if (count == 0)
        nodes = [];
      end
    end

%%%
```

```
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
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  %%%%
```

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('WindowStyle', '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('WindowStyle', '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 ^

```
function [hcomponent, hcontainer] = ltpda_javacomponent(component, position, parent)
% 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('WindowStyle', '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('WindowStyle', '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 $
%

if (usejavacomponent == 0)
    err.message = 'JAVACOMPONENT is not supported on this platform';
    err.identifier = 'MATLAB:javacomponent:FeatureNotSupported';
    error(err);
end

if ~isempty(nargchk(1,3,nargin))
    error('MATLAB:javacomponent',usage);
end

if nargin < 3
    parent = gcf;
end

if nargin < 2
    position = [20 20 60 20];
end

parentIsFigure = false;
hParent = handle(parent);
if ( isa(hParent, 'figure') || ...
     isa(hParent, 'uicontainer') || ...
     isa(hParent, 'uiflowcontainer') || ...
     isa(hParent, 'uigridcontainer'))
    parentIsFigure = true;
    peer = get(ancestor(parent,'figure'),'JavaFrame');
elseif isa(hParent, 'uitoolbar')
    peer = get(parent,'JavaContainer');
    if isempty(peer)
        drawnow;
        peer = get(parent,'JavaContainer');
    end
else
    error('MATLAB:javacomponent:InvalidParentHandle', 'Invalid parent handle\n%s',
usage)
end

if isempty(peer)
    error('MATLAB:javacomponent:JavaFigsNotEnabled', 'Java figures are not enabled')
end
```

```

hUicontainer = [];
hgp = [];
returnContainer = 1;

if ischar(component)
    component = awtcreate(component);
end

% promote the component to a handle object first.
% It seems once a java object is cast to a handle, you cannot get another
% handle with 'callbackproperties'. So, do want you need outright and use
% the same handle in the subsequent code.
if ~isjava(component)
    component =java(component);
end
hcomponent  = handle(component,'callbackProperties');

if nargin == 1
    hgp = handle(peer.addchild(component));
    % parent must be a figure, we default to gcf upstairs
    createPanel;
    hgp.setUIContainer(hUicontainer);
else
    if parentIsFigure
        if isnumeric(position)
            if isempty(position)
                position = [20 20 60 20];
            end
            % numeric position is not set here, rely on the uicontainer
            % listeners below.
            hgp = handle(peer.addchild(component));
            createPanel;
            hgp.setUIContainer(hUicontainer);
        elseif ...
                isequal(char(position),char(java.awt.BorderLayout.NORTH)) || ...
                isequal(char(position),char(java.awt.BorderLayout.SOUTH)) || ...
                isequal(char(position),char(java.awt.BorderLayout.EAST))  || ...
                isequal(char(position),char(java.awt.BorderLayout.WEST))
            hgp = handle(peer.addchild(component, position));
            returnContainer = 0;
            createPanel;
        else
            error('MATLAB:javacomponent:InvalidPosition', 'Invalid component position\n%s', usage)
        end
    else
        % Adding component to the toolbar.
        % component position is ignored for now
        peer.add(component);
        hUicontainer = parent; % toolbar.
        handles = getappdata(hUicontainer, 'childhandles');
        handles = [handles, hcomponent];
        setappdata(hUicontainer, 'childhandles', handles);
    end

    % make sure the component is on the screen so the
    % caller can interact with it right now.
    % drawnow;
end

configureComponent;

if (returnContainer == 1)
    hcontainer = hUicontainer;
else
    hcontainer = [];
end

    function createPanel
        % add delete listener
        hUicontainer = hgjavacomponent('Parent', parent, 'Units', 'Pixels');

        set(hUicontainer, 'UserData', char(component.getClass.getName)); % For findobj queries.
        if isa(java(hgp), 'com.mathworks.hg.peer.FigureChild')
            set(hUicontainer, 'FigureChild', hgp);
        end
        if isa(java(hcomponent), 'javax.swing.JComponent')
            % force component to be opaque if it's a JComponent. This prevents
            % lightweight component from showing the figure background (which
            % may never get a paint event)
            hcomponent.setOpaque(true);
        end
        set(hUicontainer, 'JavaPeer', hcomponent);

```

```
        if (returnContainer == 1)
            % add resize listener to parent (parent must be a figure or this dies quietly)
            % this is for normalized units
            addlistener(hUicontainer, 'PixelBounds', 'PropertyPostSet', @handleResize);

            % add visible listener
            addlistener(hUicontainer, 'Visible', 'PropertyPostSet', @handleVisible);

            % add parent listener
            addlistener(hUicontainer, 'Parent', 'PropertyPreSet', @handlePreParent);

            % add parent listener
            addlistener(hUicontainer, 'Parent', 'PropertyPostSet', @handlePostParent);

            % force though 1st resize event
            set(hUicontainer,'Position', position);
        else
            % For the BorderLayout components, we dont really want the
            % hUicontainer to show. But, it provides a nice place for cleanup.
            set(hUicontainer,'Visible', 'off', 'Handlevisibility', 'off');
            % Set position out of the figure to work around a current bug
            % due to which invisible uicontainers show up when renderer is
            % OpenGL (G.
            set(hUicontainer, 'Position', [1 1 0.01 0.01]);
        end

        if isa(component,'com.mathworks.hg.peer.FigureChild')
            component.setUIContainer(hUicontainer);
        end

        function handleResize(obj, evd) %#ok - mlint
            hgp.setPixelPosition(getpixelposition(hUicontainer, true));
        end

        function handleVisible(obj, evd) %#ok - mlint
            hgp.setVisible(strcmp(get(hUicontainer,'Visible'),'on'))
        end

        function handlePreParent(obj, evd) %#ok - mlint
            oldfig = ancestor(parent, 'figure');
            newfig = ancestor(evd.NewValue, 'figure');
            if ~isempty(newfig) && ~isequal(oldfig, newfig)
                peer = get(oldfig,'JavaFrame');
                peer.remove(component);
            end
        end

        function handlePostParent(obj, evd) %#ok - mlint
            oldfig = ancestor(parent, 'figure');
            newfig = ancestor(evd.NewValue, 'figure');
            if ~isempty(newfig) && ~isequal(oldfig, newfig)
                peer = get(newfig,'JavaFrame');
                hgp= handle(peer.addchild(component));
                if isa(java(hgp), 'com.mathworks.hg.peer.FigureChild')
                    % used by the uicontainer C-code
                    setappdata(hUicontainer, 'FigureChild', java(hgp));
                end
                parent = newfig;
            end
            hgp.setPixelPosition(getpixelposition(hUicontainer, true));
        end
    end

    function configureComponent
        set(hUicontainer,'DeleteFcn', {@containerDelete, hcomponent});
        addlistener(hcomponent, 'ObjectBeingDestroyed', {@componentDelete, hUicontainer, parentIsFigure});
    end

end

function containerDelete(obj, evd, hc) %#ok - mlint
    if isa(handle(obj), 'uitoolbar')
        childHandles = getappdata(obj, 'childhandles');
        delete(childHandles);
    else
        if ishandle(hc)
            delete(hc);
        end
    end
end

function componentDelete(obj, evd, hUicontainer, parentIsFigure) %#ok - mlint
if (parentIsFigure)
    % This java component is always deleted before hUicontainer. It is
```

```
    % ensured by calling component deletion in function containerDelete.
    % hUicontainer becomes invalid when delete(hUicontainer) below is run.
    parent = ancestor(hUicontainer,'figure');
    peer = get(parent, 'JavaFrame');

    if any(ishandle(obj))
        removeobj = java(obj);
        if ~isempty(get(hUicontainer,'FigureChild'))
            removeobj  = get(hUicontainer,'FigureChild');
        end
        peer.remove(removeobj);
    end

    % delete container if it exists
    if any(ishandle(hUicontainer))
        delete(hUicontainer);
    end
else
    parent = hUicontainer; % toolbar
    if ~ishandle(parent) || ~ishandle(obj)
        % The toolbar parent or the component has been deleted. Bail out.
        % Toolbar clears all javacomponents after itself.
        return;
    end

    peer = get(parent, 'JavaContainer');
    if ~isempty(peer)
        peer.remove(java(obj));
    end
end
end


function str=usage
str = [sprintf('\n') ...
    'WARNING: This feature is not supported in MATLAB and the API and ' sprintf('\n') ...
    'functionality may change in a future release. ' sprintf('\n') ...
];
%{
    'usage: javacomponent(hJavaComponent, position, parent) ' sprintf('\n') ...
    '- position can be [left bottom width height] in pixels ' sprintf('\n') ...
    '  or the string North, South, East, or West' sprintf('\n') ...
    '- parent can be a figure or a uitoolbar handle.'];
%}
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function [tree, container] = ltpda_uitree(varargin)
% 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.
```

```
0014 %
0015 %   UITREE(figurehandle, ...) creates a uitree object in the figure
0016 %   window specified by the figurehandle.
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
```

```
%
% $Id: ltpda_uitree.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
%

%   Release: R14. This feature will not work in previous versions of MATLAB.

%% Setup and P-V parsing.

error(javachk('awt'));
error(nargoutchk(0, 2, nargout));

fig = [];
numargs = nargin;

if (nargin > 0 && isscalar(varargin{1}) && ishandle(varargin{1}))
    if ~isa(handle(varargin{1}), 'figure')
        error('MATLAB:uitree:InvalidFigureHandle', 'Unrecognized parameter.');
    end
    fig = varargin{1};
    varargin = varargin(2:end);
    numargs = numargs - 1;
end

% RootFound = false;
root   = [];
expfcn = [];
selfcn = [];
pos    = [];
% parent = [];

if (numargs == 1)
    error('MATLAB:uitree:InvalidNumInputs', 'Unrecognized parameter.');
end

for i = 1:2:numargs-1
    if ~ischar(varargin{i})
        error('MALTAB:uitree:UnrecognizedParameter', 'Unrecognized parameter.');

    end
    switch lower(varargin{i})
        case 'root'
            root = varargin{i+1};
        case 'expandfcn'
            expfcn = varargin{i+1};
        case 'selectionchangefcn'
            selfcn = varargin{i+1};
        case 'parent'
            if ishandle(varargin{i+1})
                f = varargin{i+1};
                if isa(handle(f), 'figure')
                    fig = f;
                end
            end
        case 'position'
            p = varargin{i+1};
            if isnumeric(p) && (length(p) == 4)
                pos = p;
            end
        otherwise
            error('MALTAB:uitree:UnknownParameter', ['Unrecognized parameter: ',
varargin{i}]);
    end
end

if isempty(expfcn)
    [root, expfcn] = processNode(root);
else
    root = processNode(root);
end

tree_h = com.mathworks.hg.peer.UITreePeer;
tree_h.setRoot(root);

if isempty(fig)
    fig = gcf;
end

if isempty(pos)
    figpos = get(fig, 'Position');
    pos =  [0 0 min(figpos(3), 200) figpos(4)];
end

% pass the figure child in, let javacomponent introspect
[obj, container] = javacomponent(tree_h, pos, fig);
% javacomponent returns a UDD handle for the java component passed in.
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.
```

```
    try
        iconpath = [matlabroot, '/toolbox/matlab/icons/foldericon.gif'];
        node = uitreenode(root, root, iconpath, ~isdir(root));
    catch
        node = [];
    end
    expfcn = @dirBrowser;
else
    node = [];
end

end

%% -------------------------------------------------------
function nodes = hgBrowser(tree, value)                                %#ok

try
    count = 0;
    parent = handle(value);
    ch = parent.children;

    for i=1:length(ch)
        count = count+1;
        nodes(count) = uitreenode(handle(ch(i)), get(ch(i), 'Type'), [], ...
            isempty(get(ch(i), 'children')));
    end
catch
    error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not recognized. You may need to ', ...
        'define an ExpandFcn for the nodes.']);
end

if (count == 0)
    nodes = [];
end

end

%% -------------------------------------------------------
function nodes = mdlBrowser(tree, value)                               %#ok

try
    count = 0;
    parent = handle(get_param(value,'Handle'));
    ch = parent.getHierarchicalChildren;

    for i=1:length(ch)
        if isempty(findstr(class(ch(i)),'SubSystem'))
            % not a subsystem
        else
            % is a subsystem
            count = count+1;
            descr = get(ch(i),'Name');
            isleaf = true;
            cch =  ch(i).getHierarchicalChildren;
            if ~isempty(cch)
                for j = 1:length(cch)
                    if ~isempty(findstr(class(cch(j)),'SubSystem'))
                        isleaf = false;
                        break;
                    end
                end
            end
            nodes(count) = uitreenode([value '/' descr], descr, [], ...
                isleaf);
        end
    end
catch
    error('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not recognized. You may need to ', ...
        'define an ExpandFcn for the nodes.']);
end

if (count == 0)
    nodes = [];
end

end


%% -------------------------------------------------------
function nodes = dirBrowser(tree, value)                               %#ok

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('MATLAB:uitree:UnknownNodeType', ['The uitree node type is not recognized. You may need to ', ...
        'define an ExpandFcn for the nodes.']);
end

if (count == 0)
    nodes = [];
end

end

%% -----------------------------------------------------
function yesno = ismodel(input)
yesno = false;

try
    get_param(input,'handle');
    yesno = true;
catch
end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpda_uitreenode

PURPOSE ^

WARNING: This feature is not supported in MATLAB

SYNOPSIS ^

function node = ltpda_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 ^

```
function node = ltpda_uitreenode(value, string, icon, isLeaf)
% 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 $
%

import com.mathworks.hg.peer.UITreeNode;
node = handle(UITreeNode(value, string, icon, isLeaf));
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_constructo
- Callback executed when user selects a parameter set
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = ltpda_constructor_helper(varargin)

% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% Check if I exist already
id = findobj('Tag', 'LTPDAconstructor_helper');
if ~isempty(id)
  figure(id)
  return
end
```

```
%% Some initial setup

Gproperties.Gcol    = [240 240 240]/255;
Gproperties.Gwidth  = 500;
Gproperties.Gheight = 300;
Gproperties.Gborder = 10;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');

Gproperties.Screen   = get(0,'screensize');
Gproperties.Gposition = [150 ...
  150 ...
  Gproperties.Gwidth...
  Gproperties.Gheight];

%  Initialize and hide the GUI as it is being constructed.
mainfig = figure('Name', 'LTPDA Constructor Helper',...
  'NumberTitle', 'off',...
  'Visible','off',...
  'Position',Gproperties.Gposition,...
  'Color', Gproperties.Gcol,...
  'Resize', 'off',...
  'ToolBar',      'none',            ...
  'NextPlot',      'new',            ...
  'MenuBar', 'none',...
  'Tag', 'LTPDAconstructor_helper');

% Set mainfig callbacks
set(mainfig, 'CloseRequestFcn', {@ltpda_constructor_helper_close, mainfig});

% Set Application data
setappdata(mainfig, 'Gproperties', Gproperties);


%% GUI parts

%--- Class selection
% text field
sth = uicontrol(mainfig,'Style','text',...
  'String','Class',...
  'HorizontalAlignment', 'left', ...
  'BackgroundColor', Gproperties.Gcol, ...
  'Position',[10 Gproperties.Gheight-40 50 25]);

% pop-up dialog
pmh = uicontrol(mainfig,'Style','popupmenu',...
  'String', utils.helper.ltpda_userclasses,...
  'Value', 1, ...
  'BackgroundColor', Gproperties.Gcol, ...
  'Tag', 'LTPDA_constructor_helper_class', ...
  'Callback', {'ltpda_constructor_helper_class', mainfig}, ...
  'Position',[90 Gproperties.Gheight-40 120 25]);

%--- Constructor set list


% text field
sth = uicontrol(mainfig,'Style','text',...
  'String','Sets',...
  'HorizontalAlignment', 'left', ...
  'BackgroundColor', Gproperties.Gcol, ...
  'Position',[10 Gproperties.Gheight-80 50 25]);

% Set list
lbh = uicontrol(mainfig,'Style','listbox',...
                'String', '',...
                'Value', 1, ...
                'BackgroundColor', 'w', ...
                'Tag', 'LTPDA_constructor_helper_sets', ...
                'Callback', {'ltpda_constructor_helper_sets', mainfig}, ...
                'Position',[10 Gproperties.Gheight-270 200 200]);

% Constructor view
eth = uicontrol(mainfig,'Style','edit',...
                'String','',...
                'BackgroundColor', 'w', ...
                'Max', 100, ...
                'HorizontalAlignment', 'left', ...
                'FontSize', 14, ...
                'Tag', 'LTPDA_constructor_helper_constructor', ...
                'Position',[240 Gproperties.Gheight-210 250 200]);

% Variable name
sth = uicontrol(mainfig,'Style','text',...
  'String','Variable',...
  'HorizontalAlignment', 'left', ...
```

```
  'BackgroundColor', Gproperties.Gcol, ...
  'Position',[240 Gproperties.Gheight-260 50 25]);

eth = uicontrol(mainfig,'Style','edit',...
                'String','obj',...
                'BackgroundColor', 'w', ...
                'HorizontalAlignment', 'center', ...
                'Tag', 'LTPDA_constructor_helper_var', ...
                'Position',[300 Gproperties.Gheight-260 50 25]);


%  Build btn
pbh = uicontrol(mainfig,'Style','pushbutton','String','Build',...
  'Position',[360  Gproperties.Gheight-260 60 25],...
  'Tag', 'LTPDA_constructor_helper_buildBtn');
set(pbh, 'Callback', {'ltpda_constructor_helper_build'});

%% Start the GUI

% Make the GUI visible.
set(mainfig,'Visible','on')


%% Callbacks

  %--- Close
  function ltpda_constructor_helper_close(varargin)
    disp('* Goodbye from the LTPDA Constructor Helper *')
    delete(varargin{end})
  end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_constructor_helper_build(varargin)
% Callback for the build constructor gui

mainfig = findobj('Tag', 'LTPDAconstructor_helper');

% get object name
varh = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_var');
varname = get(varh, 'String');

% get selected class
clh   = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
cltxt = get(clh, 'String');
cl    = cltxt{get(clh, 'Value')};

% get base workspace variables
ws_vars = evalin('base','who');

% check if this one is there or not
if ismember(varname, ws_vars)
  j = 1;
  vname = varname;
  while ismember(vname, ws_vars)
    vname = sprintf('%s_%03d', varname, j);
    j = j + 1;
  end
else
  vname = varname;
end

disp(sprintf('** exporting to %s', vname));

% Get constructor string
ch   = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_constructor');
cstr = get(ch, 'String');

cmd = sprintf('%s = %s;', vname, cstr);
evalin('base', cmd);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_constructor_helper_buildParams(varargin)
% Build parameter panel on the main figure


mainfig = varargin{1};
pl      = varargin{2};

% get selected class
clh   = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
cltxt = get(clh, 'String');
cl    = cltxt{get(clh, 'Value')};

% Convert plist to string
str = string(pl);

ch = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_constructor');

set(ch, 'String', sprintf('%s(%s)', cl, str));


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_constructor_helper_class(varargin)
% Callback executed when user selects a class

mainfig = findobj('Tag', 'LTPDAconstructor_helper');

% get selected class
clh   = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
cltxt = get(clh, 'String');
cl    = cltxt{get(clh, 'Value')};

lh   = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_sets');


% Get the info
eval(sprintf('ii = %s.getInfo;', cl));
sets = ii.sets;
% eval(sprintf('sets = %s(%s, ''Params'');', cl, cl));

% Fill set list
if isa(sets, 'plist')
  setappdata(mainfig, 'plist', sets);
  set(lh, 'Value', 1);
  set(lh, 'String', {'Default'});
else
  set(lh, 'Value', 1);
  set(lh, 'String', sets);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_constructor_helper_sets(varargin)

% Callback executed when user selects a parameter set

mainfig = findobj('Tag', 'LTPDAconstructor_helper');

% get selected class
clh   = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_class');
cltxt = get(clh, 'String');
cl    = cltxt{get(clh, 'Value')};

% get selected set
lh   = findobj(mainfig, 'Tag', 'LTPDA_constructor_helper_sets');
sets = get(lh, 'String');
set  = sets{get(lh, 'Value')};

if strcmp(set, 'Default')
  pl = getappdata(mainfig, 'plist');
else
  cmd = sprintf('pl = %s.getInfo(''%s'', ''%s'').plists;', cl, cl, set);
  eval(cmd);
end

ltpda_constructor_helper_buildParams(mainfig, pl);

```

*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_arrmult">m/gui/gltpda/g_arrmult</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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function buildLibrary(varargin)
% Function to update the library with all the methods functions.
%  $Id: buildLibrary.m,v 1.9 2008/08/04 20:07:33 nicola Exp $

%   clear all
  bdclose('all');
  disp('=====================================================')
  disp('Updating the LTPDA Library...');

% Retrieve the list of all functions and methods:
ltpda_versions

% This creates a variable named 'versions', so structured:
% | 1.filename | 2.functionName | 3.version | 4.date | 5.hour | 6.author | 7.folder |
8.hash | 9.category | 10.class |
% [The 'class' is set to None if it's not a method]

  origSize = size(versions,1); %#ok<NODEF>
  disp(['The overall number of LTPDA functions is ',num2str(origSize),'.'])


% ========================================================================
%%              Remove functions not meant for the library:
% ========================================================================

   % Remove all test functions:
     for i=origSize:-1:1
         if numel(versions{i,2})>4 && strcmpi(versions{i,2}(1:5),'test_')
             versions(i,:)=[];
         end
     end
     newSize = size(versions,1);
     disp(['--- ',num2str(origSize-newSize),' test functions ignored'])
     origSize = newSize;

   % Remove all Internal functions:
     for i=origSize:-1:1
         if numel(versions{i,9})>7 && strcmpi(versions{i,9}(1:8),'Internal')
             versions(i,:)=[];
         end
     end
     newSize = size(versions,1);
     disp(['--- ',num2str(origSize-newSize),' Internal functions ignored'])
     origSize = newSize;

```

```
    % Remove all private functions:
      for i=origSize:-1:1
          x = numel(versions{i,10});
          if x>8 && strcmpi(versions{i,10}(x-7:x),'/private')
              versions(i,:)=[];
          end
      end
      newSize = size(versions,1);
      disp(['--- ',num2str(origSize-newSize),' private functions ignored'])
      origSize = newSize;

    % Remove all remaining examples:
      for i=origSize:-1:1
          x = numel(versions{i,7});
          if x>10 && strcmpi(versions{i,7}(x-9:x),'/examples/')
              versions(i,:)=[];
          end
      end
      newSize = size(versions,1);
      disp(['--- ',num2str(origSize-newSize),' examples ignored'])
      origSize = newSize;

    % Remove templates:
      for i=origSize:-1:1
          funcName = versions{i,2};
          x = numel(funcName);
          if x>9 && strcmpi(funcName(x-8:x),'_template')
              versions(i,:)=[];
          end
      end
      newSize = size(versions,1);
      disp(['--- ',num2str(origSize-newSize),' templates ignored'])
      origSize = newSize;

    % Remove all uncategorized functions:
      for i=origSize:-1:1
          if isempty(versions{i,9})
              versions(i,:)=[];
          end
      end
      newSize = size(versions,1);
      disp(['--- ',num2str(origSize-newSize),' uncategorized functions ignored'])
      origSize = newSize;

    % Remove all functions with no class:
      for i=origSize:-1:1
          if strcmp(versions{i,10},'None')
              versions(i,:)=[];
          end
      end
      newSize = size(versions,1);
      disp(['--- ',num2str(origSize-newSize),' functions with class ''None'' ignored'])
      origSize = newSize;

    % Remove all functions with non user classes:
      for i=origSize:-1:1
          if numel(versions{i,10})>5 && strncmpi(versions{i,10},'ltpda_',6)
             versions(i,:)=[];
          elseif numel(versions{i,10})>3 && strncmpi(versions{i,10},'data',4)
             versions(i,:)=[];
          end
      end
      newSize = size(versions,1);
      disp(['--- ',num2str(origSize-newSize),' functions with class ''ltpda_***''
ignored'])
      origSize = newSize;
% ==========================================================================
%%                      Prepare the library for update:
% ==========================================================================

     disp(['The number of LTPDA functions for the library is ',num2str(origSize),'.'])
        numbFunctions = 0;
        funcCategories = {};
        for i=1:origSize
            if numel(versions{i,10})>3 && strcmpi(versions{i,10}(1:4),'None')
                numbFunctions = numbFunctions +1;
                if ~ismember(versions{i,9},funcCategories)
                    funcCategories = [funcCategories,versions{i,9}];
                end
            end
        end
      % These are the root level subsystems:
        funcCategories = [funcCategories,'Methods'];
     disp([num2str(numbFunctions),' are functions, ',num2str(origSize-numbFunctions),'
are methods.'])
     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')
```

```
                    blockH = add_block('libraryBase/constructor',['ltpda_library/Methods/',currClass,'/',currCateg,'/',currMethod]);
                elseif strcmp(currCateg,'Output')
                    blockH = add_block('libraryBase/output',['ltpda_library/Methods/',currClass,'/',currCateg,'/',currMethod]);
                elseif ~strcmp(currMethod,'Contents')
                    blockH = add_block('libraryBase/function',['ltpda_library/Methods/',currClass,'/',currCateg,'/',currMethod]);
                end

            % Cycle to set proper position:
                x=1;y=1;
                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)]))
                    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)]);
                    if (strcmp(aaa{1},['ltpda_library/Methods/',currClass,'/',currCateg]) && numel(aaa)==1), break; end
                    y=y+1;
                    if y==5, x=x+1; y=1; end
                end
                set(blockH,'Position',[30+70*(y-1) , 30+70*(x-1) , 80+70*(y-1) , 60+70*(x-1)])

                set(blockH,'LinkStatus','inactive')
                set(blockH,'MaskDisplay',['disp(''',currMethod,''')'])
                set(blockH,'Tag',['method ',currClass])
                set(blockH,'Description','')
                maskText = eval(['help(''',currClass,'/',currMethod,''');']);
                maskText = strtok(maskText,'%%');
                set(blockH,'MaskDescription',maskText);
                set(blockH,'OpenFcn','ltpdagui(''callback'',1,''redraw'',2);if (~strcmp(bdroot,''ltpda_library'') && ~isempty(findobj(''Tag'',''LTPDAGUI''))), figure(findobj(''Tag'',''LTPDAGUI'')); end');
                childpath = find_system(blockH,'LookUnderMasks','all','BlockType','M-S-Function');
                set(childpath,'Tag',currMethod)
                set(childpath,'Name',currMethod)
                set(childpath,'Description','')
    end

    % Setting proper position and size for subsystem folders in the root:
      subsystems = find_system(bdroot,'LookUnderMasks','all','SearchDepth',1,'BlockType','SubSystem');
      x=1; y=1;
      for i=1:numel(subsystems)
          set_param(subsystems{i},'Position',[30+100*(y-1) , 30+100*(x-1) , 95+100*(y-1) , 90+100*(x-1)])
          y=y+1;
          if y==4, x=x+1; y=1; end
      end

      close_system libraryBase.mdl

    % Update the annotation:
      annotationHandle = find_system(bdroot,'FindAll','on','SearchDepth',1,'type','annotation');
      caption1 = 'LTPDA Library';
      caption2 = ['v3.0 - ',datestr(now,1)];
      caption = strvcat(caption1,caption2); %#ok<VCAT>
      set_param(annotationHandle, 'HorizontalAlignment','center','Text',caption)

      warning off all
      save_system ltpda_library.mdl
      warning on all
      close_system ltpda_library.mdl

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

g_arrmult

PURPOSE ^

This is the automatic function wrapper

SYNOPSIS ^

function g_arrmult(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_arrmult.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 ^

```
function g_arrmult(block)

%             This is the automatic function wrapper
% ========================================================================
```

```
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_arrmult.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('=========================================================')
```

```
    disp('Something''s wrong. Please check ltpdaarrmult.m function')
    disp('=========================================================')
    return
end

  outdata = LTPDAinvar{block.InputPort(1).Data(1)} .*
LTPDAinvar{block.InputPort(1).Data(2)};
  outdata = num2cell(outdata);

  lineHandles = get_param(get_param(gcbh,'Parent'),'LineHandles');
  signalName = get(lineHandles.Outport,'Name');
  if ~isempty(signalName) && numel(outdata)==1
      outdata{1} = setnh(outdata{1},'name',signalName);
  end
  xx = size(LTPDAinvar,1);
  LTPDAinvar(xx+1,:) = [outdata,0];
  block.OutputPort(1).Data = xx+1;

%endfunction
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function g_cleaninput(block)

%             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 $

setup(block);

%%
function setup(block)

  %% Register dialog parameter: none, because they're retrieved directly
  %% from the memory. This will prevent the user to modify the parameters
  %% outside the proper parameters panel:
  block.NumDialogPrms = 0;

  %% Register number of input and output ports
  block.NumInputPorts  = 2;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

```

```
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.SetAccelRunOnTLC(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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 AO(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 ^

```
function g_copyparams(block)

%             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_copyparams.m,v 1.1 2008/03/01 13:43:20 nicola Exp $

setup(block);

%%
function setup(block)

  %% Register dialog parameter: none, because they're retrieved directly
  %% from the memory. This will prevent the user to modify the parameters
  %% outside the proper parameters panel:
  block.NumDialogPrms = 0;

  %% Register number of input and output ports
  block.NumInputPorts  = 2;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically inherited.
  block.SetPreCompInpPortInfoToDynamic;
```

```
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).DirectFeedthrough = true;
  block.InputPort(1).DatatypeID = 0;
  block.InputPort(1).Complexity = 0;
% block.InputPort(1).Dimensions = 2;
  block.InputPort(2).DirectFeedthrough = true;
  block.InputPort(2).DatatypeID = 0;
  block.InputPort(2).Complexity = 0;
% block.InputPort(2).Dimensions = 2;
  block.OutputPort(1).DatatypeID = 0;
  block.OutputPort(1).Complexity = 0;
% block.OutputPort(1).Dimensions = 1;
  block.SampleTimes = [0 0];
  block.SetAccelRunOnTLC(false);

  %% Register methods
%   block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
  block.RegBlockMethod('SetInputPortDimensions',  @SetInpPortDims);
  block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
  block.RegBlockMethod('Outputs',                  @Outputs);

%   function SetInpPortFrameData(block, idx, fd)
%   block.InputPort(1).SamplingMode = fd;
%   block.OutputPort(1).SamplingMode = fd;

  function SetInpPortDims(block, idx, di)
  block.InputPort(idx).Dimensions = di;

  function SetOutPortDims(block, idx, di)
  block.OutputPort(idx).Dimensions = di;

%%
function Outputs(block)

% currBlk=get(gcbh,'Parent');
  currBlkConnect=get_param(get(gcbh,'Parent'),'PortConnectivity');
  parentBlkHandle=currBlkConnect(2).SrcBlock;
  childBlkHandle=currBlkConnect(3).DstBlock;
  param2copy=get_param(parentBlkHandle,'Description');
  set_param(childBlkHandle,'Description',param2copy);
  annotation=get_param(parentBlkHandle,'AttributesFormatString');
  set_param(childBlkHandle,'AttributesFormatString',annotation);

  block.OutputPort(1).Data = block.InputPort(1).Data;

%endfunction

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function g_iplot(block)

%             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 $

setup(block);

%%
function setup(block)
global LTPDAinvar;

  %% Register dialog parameter: none, because they're retrieved directly
  %% from the memory. This will prevent the user to modify the parameters
  %% outside the proper parameters panel:
  block.NumDialogPrms = 0;

  %% Register number of input and output ports
  block.NumInputPorts  = 1;
  block.NumOutputPorts = 1;

```

```
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.SetAccelRunOnTLC(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{b

0071     case 4
0072
iplot(LTPDAinvar{block.InputPort(1).Data(1)},LTPDAinvar{block.InputPort(1).Data(2)},LTPDAinvar{b

0073 end
0074
0075 % %=====================================================================
0076
0077    block.OutputPort(1).Data = block.InputPort(1).Data;
0078
0079 %endfunction
0080
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function g_pl2ao(block)
% For the block plist2AO : let the user to build a plist and convert it
% into an AO in memory at the time of execution.

%  $Id: g_pl2ao.m,v 1.1 2008/03/01 13:43:20 nicola Exp $Revision: 1.1 $

setup(block);

%%
function setup(block)

  %% Register dialog parameter: none, because they're retrieved directly
  %% from the memory. This will prevent the user to modify the parameters
  %% outside the proper parameters panel:
  block.NumDialogPrms = 0;

  %% Register number of input and output ports
  block.NumInputPorts  = 0;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically inherited.
  block.SetPreCompOutPortInfoToDynamic;

  block.OutputPort(1).DatatypeID = 0;
  block.OutputPort(1).Complexity = 0;
  block.OutputPort(1).Dimensions = 1;
  block.OutputPort(1).SamplingMode = 'sample';
  block.SampleTimes = [0 0];
  block.SetAccelRunOnTLC(false);

  %% Register methods
% block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
  block.RegBlockMethod('Outputs',                 @Outputs);

% function SetOutPortDims(block, idx, di)
% block.OutputPort(idx).Dimensions = 1;

```

```
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 ^

```
function g_plot(block)

%              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 $

setup(block);

%%
function setup(block)
global AnOb;

  %% Register dialog parameter: none, because they're retrieved directly
  %% from the memory. This will prevent the user to modify the parameters
  %% outside the proper parameters panel:
  block.NumDialogPrms = 0;

  %% Register number of input and output ports
  block.NumInputPorts  = 1;
  block.NumOutputPorts = 1;

```

```
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.SetAccelRunOnTLC(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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function g_resp(block)

%  $Id: g_resp.m,v 1.1 2008/03/01 13:43:20 nicola Exp $

setup(block);

%%
function setup(block)

  %% Register dialog parameter: none, because they're retrieved directly
  %% from the memory. This will prevent the user to modify the parameters
  %% outside the proper parameters panel:
  block.NumDialogPrms = 0;

  %% Register number of input and output ports
  block.NumInputPorts  = 1;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically inherited.
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).DirectFeedthrough = true;
  block.InputPort(1).DatatypeID = 0;
  block.InputPort(1).Complexity = 0;
  block.InputPort(1).SamplingMode = 'sample';
  block.OutputPort(1).DatatypeID = 0;
  block.OutputPort(1).Complexity = 0;
  block.OutputPort(1).Dimensions = 1;
  block.OutputPort(1).SamplingMode = 'sample';
  block.SampleTimes = [0 0];
  block.SetAccelRunOnTLC(false);


```

```
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;
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function g_saveresult(block)

%             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 $

setup(block);

%%
function setup(block)

  %% Register dialog parameter: none, because they're retrieved directly
  %% from the memory. This will prevent the user to modify the parameters
  %% outside the proper parameters panel:
  block.NumDialogPrms = 0;

  %% Register number of input and output ports
  block.NumInputPorts  = 1;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically inherited.
```

```
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).DirectFeedthrough = true;
  block.InputPort(1).DatatypeID = 0;
  block.InputPort(1).Complexity = 0;
% block.InputPort(1).Dimensions = 2;
  block.OutputPort(1).DatatypeID = 0;
  block.OutputPort(1).Complexity = 0;
% block.OutputPort(1).Dimensions = 1;
  block.SampleTimes = [0 0];
  block.SetAccelRunOnTLC(false);

  %% Register methods
  block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
  block.RegBlockMethod('SetInputPortDimensions',  @SetInpPortDims);
  block.RegBlockMethod('SetOutputPortDimensions', @SetOutPortDims);
  block.RegBlockMethod('Outputs',                  @Outputs);

  function SetInpPortFrameData(block, idx, fd)
  block.InputPort(1).SamplingMode = fd;
  block.OutputPort(1).SamplingMode = fd;

  function SetInpPortDims(block, idx, di)
  block.InputPort(idx).Dimensions = di;

  function SetOutPortDims(block, idx, di)
  block.OutputPort(idx).Dimensions = di;


%%
function Outputs(block)
global LTPDAinvar LTPDAoutvar

% Check if the user wants to stop the execution:
lastChar = get(findobj('Name','LTPDA Progress Bar'),'CurrentCharacter');
if ~isempty(lastChar) && strcmp(lastChar,'x')
    set_param(bdroot, 'SimulationCommand', 'stop')
    return
end

y = numel(block.InputPort(1).Data);
for j=1:y
    yy = numel(LTPDAinvar{block.InputPort(1).Data(j)});
    if yy >1
        for jj=1:yy
            LTPDAoutvar = [LTPDAoutvar;{LTPDAinvar{block.InputPort(1).Data(j)}(jj)}];
        end
    else
        LTPDAoutvar = [LTPDAoutvar;{LTPDAinvar{block.InputPort(1).Data(j)}}];
    end
end

% %=========================================================================

block.OutputPort(1).Data = block.InputPort(1).Data;

%endfunction

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function [varargout]=loginLTPDA(varargin)
   % 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 $

   Login    = varargin{1};
   password = varargin{2};
   for i=1:numel(password), passwordHidden(i)='*'; end
   if ~exist('passwordHidden','var'), passwordHidden = ''; end

   set(0,'Units','pixels')
   Screen = get(0,'screensize');
   loginSize = [230 130];
   Position = [(Screen(3)-loginSize(1))/2 (Screen(4)-loginSize(2))/2 loginSize];


   % Create the GUI
   gui.main = dialog('HandleVisibility','on',...
      'Color','white',...
      'IntegerHandle','off',...
      'Menubar','none',...
      'Name','LTPDA Login',...
      'NumberTitle','off',...
      'Name','Login',...
      'Tag','logindlg',...
      'Units','pixels',...
      'Userdata','logindlg',...
      'Position',Position);

   set(gui.main,'Closerequestfcn',{@Cancel,gui.main},'Keypressfcn',{@Escape,gui.main})

   % Texts
   gui.login_text = uicontrol(gui.main,'Style','text','BackgroundColor','white','FontSize',8,'HorizontalAlign','center','Units','pixels','String','Login','Position',[0 105 Position(3) 20]);
   gui.password_text = uicontrol(gui.main,'Style','text','BackgroundColor','white','FontSize',8,'HorizontalAlign','center','Units','pixels','String','Password','Position',[0 60 Position(3) 20]);

   % Edits
   gui.edit1 = uicontrol(gui.main,'Tag','userid','Style','edit','FontSize',8,'HorizontalAlign','center','BackgroundColor','white','Units','pixels','String',Login,'KeyPressfcn',{@KeyPress_Func2,gui.main},[5 85 Position(3)-10 24],'Userdata',0);
   gui.edit2 = uicontrol(gui.main,'Tag','password','Style','edit','FontSize',8,'HorizontalAlign','center','BackgroundColor','white','Units','pixels','String',passwordHidden,'Position',[5 40 Position(3)-10 24],'KeyPressfcn',{@KeyPress_Function,gui.main},'Userdata',password);

   % Buttons
   buttonSize = [75 26];
   gui.OK = uicontrol(gui.main,'Style','push','FontSize',8,'Units','pixels','String','OK','Position',[Position(3)/2-buttonSize(1)-1 5 buttonSize],'KeyPressFcn',{@OKReturn,gui.main},'Callback',{@OK,gui.main});
   gui.Cancel = uicontrol(gui.main,'Style','push','FontSize',8,'Units','pixels','String','Cancel','Position',[Position(3)/2+1 5 buttonSize],'KeyPressFcn',{@CancelReturn,gui.main},'Callback',{@Cancel,gui.main});

   % Images
   logosize = [31,31];
   dimension = [5 3 logosize];
   logo = axes('Parent',gui.main,'Units','pixels','Position',dimension);
   image(imread('buttonyes.jpg'),'Parent',logo);
   axis(logo,'off');
   logosize = [31,31];
   dimension = [Position(3)-logosize(1)-5 3 logosize];
   logo = axes('Parent',gui.main,'Units','pixels','Position',dimension);
   image(imread('buttonno.jpg'),'Parent',logo);
   axis(logo,'off');

   setappdata(0,'logindlg',gui) % Save handle data
```

```
   setappdata(gui.main,'Check',0) % Error check setup. If Check remains 0 an empty cell array will be returned

   uicontrol(gui.edit1) % Make the first edit box active
   % uicontrol(gui.OK) % Make the OK button active

   % Pause the GUI and wait for a button to be pressed
   uiwait(gui.main)

   try if varargin{3}, welcome2LTPDA(varargin{4},varargin{5}); pause(1); end; catch end

   Check = getappdata(gui.main,'Check'); % Check to see if a button was pressed

   % Format output
   if Check == 1
      Login = get(gui.edit1,'String');
      Password = get(gui.edit2,'Userdata');

      varargout(1) = {Login};
      varargout(2) = {Password};
   else % If OK wasn't pressed output empty fields
      varargout(1) = {''};
      varargout(2) = {''};
   end

   delete(gui.main) % Close the GUI
   setappdata(0,'logindlg',[]) % Erase handles from memory

   %% Hide Password
function KeyPress_Function(h,eventdata,fig)
   % Function to replace all characters in the password edit box with
   % asterixes
   password = get(h,'Userdata');
   key = get(fig,'currentkey');

   switch key
      case 'backspace'
         password = password(1:end-1); % Delete the last character in the password
      case 'return'  % This cannot be done through callback without making tab to the same thing
         gui = getappdata(0,'logindlg');
         OK([],[],gui.main);
      case 'tab'  % Avoid tab triggering the OK button
         gui = getappdata(0,'logindlg');
         uicontrol(gui.OK);
      otherwise
         password = [password get(fig,'currentcharacter')]; % Add the typed character to the password
   end

   SizePass = size(password); % Find the number of asterixes
   if SizePass(2) > 0
      asterix(1,1:SizePass(2)) = '*'; % Create a string of asterixes the same size as the password
      set(h,'String',asterix) % Set the text in the password edit box to the asterix string
   else
      set(h,'String','')
   end

   set(h,'Userdata',password) % Store the password in its current state

function KeyPress_Func2(h,eventdata,fig)
   % To check whenever the user press Enter
   if ~isempty(get(fig,'CurrentCharacter')) && real(get(fig,'CurrentCharacter'))==13 && get(h,'UserData')==0
      setappdata(fig,'Check',1)
      uiresume(fig)
   elseif real(get(fig,'CurrentCharacter'))==13
      uicontrol(findobj(gcf,'Tag','password'))
   else
      set(h,'UserData',1)
   end

   %% Cancel
function Cancel(h,eventdata,fig)
   uiresume(fig)

   %% OK
function OK(h,eventdata,fig)
   % Set the check and resume
   setappdata(fig,'Check',1)
   uiresume(fig)

function CancelReturn(h,eventdata,fig)
   if real(get(fig,'CurrentCharacter'))==13
      uiresume(fig)
   end

   %% OK
function OKReturn(h,eventdata,fig)
   % Set the check and resume
   if real(get(fig,'CurrentCharacter'))==13
      setappdata(fig,'Check',1)
      uiresume(fig)
```

```
    end

    %% Escape
function Escape(h,eventdata,fig)
    % Close the login if the escape button is pushed and neither edit box is
    % active
    key = get(fig,'currentkey');

    if isempty(strfind(key,'escape')) == 0 && h == fig
        Cancel([],[],fig)
    end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdagui(varargin)

% ==========================================================================
%                                 LTPDA GUI
% ==========================================================================
%
%  Open the main LTPDA GUI.
%
%  $Id: ltpdagui.m,v 1.21 2008/08/30 12:27:08 nicola Exp $

warning off all
global LTPDAinvar LTPDAoutvar active labels scaleX

tabbedFigName = 'GUI Fawkes';
   screenSize = get(0,'screensize');
   scaleX     = screenSize(3)/1280;
   if scaleX<1, scaleX = 1; end
   setappdata(0, 'ltpda_gui_scalefactor',scaleX);
   guiSize = [1 1 800*scaleX 550*scaleX];
guiSize = round([(screenSize(3)-guiSize(3))/2 , (screenSize(4)-guiSize(4))/2 , guiSize(3) , guiSize(4)]);

if ~isempty(findobj('Name',tabbedFigName)) && nargin==0, return; end

if isempty(LTPDAinvar),  LTPDAinvar  = {}; end
if isempty(LTPDAoutvar), LTPDAoutvar = {}; end

userID = getappdata(0,'ltpda_repo_user');
  if isempty(userID), userID = ''; end
passwd = getappdata(0,'ltpda_repo_pass');
  if isempty(passwd), passwd = ''; end

if nargin==0, setappdata(0,'startup',1); [userID passwd] = loginLTPDA(userID,'',1,tabbedFigName,guiSize); try
cd(getappdata(0,'DefaultDataDirectory')); catch end; end
  if isempty(userID), userID = 'JohnDoe'; passwd=''; end

setappdata(0,'ltpda_repo_user',userID)
setappdata(0,'ltpda_repo_pass',passwd)

if nargin==0
    clc
    disp('Please wait: starting LTPDA GUI...')
    disp('Loading Simulink...')
    drawnow
    load_system('simulink')
    load_system('ltpda_library')
end

try stop(timerfindall); delete(timerfindall); catch end
t1 = timer('TimerFcn',@CheckTimerAlive,'ExecutionMode','fixedRate','Period',0.3);
```

```
t2 = timer('TimerFcn',@CheckTimerAlive,'ExecutionMode','fixedRate','Period',1);
start(t1)
start(t2)

%%

tag = '';
units = 'normalized';
position = [0,0,1,1];

% text color in the label:
  foregroundcolor = [0.2,0.2,0.2];
% color in the background, behind tabs:
  backgroundcolor = [0.7,0.7,1];
% background color, inside every tab:
  panelbackgroundcolor = [0.98,0.98,0.98];
% color of the small highlight line, above the selected tab
  titlehighlightcolor = [1,0.3,0];

horizontalalignment = 'center';

active = 0;
string = {'Main LTPDA panel','Partial execution','Output';'Parameters','Nested loops','';'Repository','Import from workspace','Globals'};
   nrows = size(string,1);
   if nrows>1
       for nn=1:nrows, labels{nn} = string(nn,:); end
       string = string(1,:);
   end
   ntab = numel(string);
   for jj=ntab:-1:1, ntab = ntab - strcmp(string(jj),''); end

% Cycle to retrieve parameters from the user:
  for mm=1:2:nargin
      property = lower(varargin{mm});
      value = varargin{mm+1};
      switch property
          case {'title','string'}
              string = value;
              nrows = size(string,1);
              if nrows>1
                  for nn=1:nrows, labels{nn} = string(nn,:);end
                  string = string(1,:);
              end
              ntab = numel(string);
              for jj=ntab:-1:1, ntab = ntab - strcmp(string(jj),''); end
          case {'frameposition','position'}
              position = value;
          case {'framebackgroundcolor','backgroundcolor'}
              backgroundcolor = value;
          case {'framebordertype','bordertype'}
              bordertype = value;
          case {'titleforegroundcolor','foregroundcolor'}
              foregroundcolor = value;
          case {'selecteditem','active'}
              active = value;
          case 'redraw'
              if value == 1 % main panel
                  TopBottomTabCbk([],[],1,1)
                  active = 1;
                  return
              end
              if value == 1.1 % the user clicked the Start button
                  TopBottomTabCbk([],[],1,1)
                  active = 1;
                  runSim = 1;
              end
```

```
                if value == 2 % Parameters panel
                    TopBottomTabCbk([],[],1,2)
                    active = 1;
                    return
                end
                if value == 6 % globals panel
                    TopBottomTabCbk([],[],3,3)
                    active = 1;
                    return
                end
                if value == 8 % nested loops
                    TopBottomTabCbk([],[],2,2)
                    active = 1;
                    return
                end

            case 'callback'
                if value==1 && strcmp(bdroot,'ltpda_library')
                    HelpCallback();
                    return;
                end
                if value==1 && isempty(findobj('Name',tabbedFigName))
                    return;
                end
            otherwise
                eval([property '= ' value ';']);
        end
    end

parent   = findobj('Name',tabbedFigName);
if ~isempty(parent), clf(parent); set(parent,'Resize','on','ResizeFcn',{@GUIResize,guiSize});
else parent =
figure('Position',guiSize,'Name',tabbedFigName,'Resize','on','ResizeFcn',{@GUIResize,guiSize},'NumberTitle','off','Toolbar','none','Menubar',
);
end

set(parent,'Tag','LTPDAGUI','NextPlot','new');

if ~exist('margins','var'), margins = {1,'pixels'}; end
if ~iscell(margins), margins = {margins,units}; end
if ~exist('bordertype','var'), bordertype = 'none'; end
if ~exist('titlebackgroundcolor','var'), titlebackgroundcolor = 0.95*panelbackgroundcolor; end
if ~active, active = 1; end

clear htab
htab = uipanel( ...
  'Parent',parent, ...
  'Units',units, ...
  'Position',position, ...
  'BackgroundColor',backgroundcolor, ...
  'BorderType',bordertype, ...
  'Visible','on', ...
  'Tag',tag);

  set(htab,'Units','pixels');
  tabpos = get(htab,'Position');


status = uicontainer( ...
  'Parent',htab, ...
  'BackgroundColor',backgroundcolor, ...
  'Units',margins{2}, ...
  'Position',[1,1,1,1], ...
  'Visible','off');

  set(status,'Units','pixels');
  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 ttextpos = [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',ttextpos, ...
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
```

```
  inset(1) = uicontrol( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'BackgroundColor',panelbackgroundcolor, ...
    'Style','text', ...
    'Visible','on');
  if ~isnan(titlehighlightcolor)
    inset(2) = uicontrol( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'BackgroundColor',titlehighlightcolor, ...
      'Style','text', ...
      'Visible','on');
    inset(3) = uicontrol( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'BackgroundColor',0.85*titlehighlightcolor, ...
      'Style','text', ...
      'Visible','on');
  end

% ======================================================================
%                              OTHER BUTTONS
  for xx=2:nrows
      localString = labels{xx};
      ntab = numel(localString);
      for tt=1:ntab
        titles(xx,tt) = uipanel( ...
          'Parent',htab, ...
          'Units','pixels', ...
          'BackgroundColor',titlebackgroundcolor, ...
          'BorderType','beveledout', ...
          'ButtonDownFcn',{@TopBottomTabCbk,tt,xx}, ...
          'Visible','on');

        uicontrol( ...
          'Parent',titles(xx,tt), ...
          'Units','normalized', ...
          'Position',ttextpos, ...
          'ForegroundColor',foregroundcolor, ...
          'BackgroundColor',titlebackgroundcolor, ...
          'HorizontalAlignment',horizontalalignment, ...
          'Style','text', ...
          'String',localString{tt}, ...
          'Enable','inactive', ...
          'FontSize',8*scaleX,...
          'HitTest','off',...
          'Visible','on');

     end
  end
% ======================================================================

  TopBottomTabResizeCbk();

panelTitle = get(ttext(active),'string');
setappdata(0, 'ltpda_currPanel', panel);

% ======================================================================
% ======================================================================
% ======================================================================
CheckTimerAlive()

   switch panelTitle
      case 'Main LTPDA panel'
         if exist('runSim','var') && runSim==1
            runSim = 0; %#ok<NASGU>
```

```
            pan1main(panel,1)
         else pan1main(panel)
         end
      case 'Parameters'
         pan2param(panel)
      case 'Repository'
         pan3repo(panel)
      case 'Import from workspace'
         pan4import(panel)
      case 'Output'
         pan5output(panel)
      case 'Globals'
         pan6globals(panel)
      case 'Partial execution'
         pan7partial(panel)
      case 'Nested loops'
         panXyettocome(panel)
       % pan8nested(panel)

   end

% ======================================================================
% ======================================================================
% ======================================================================


%%
    %-------------------------------------------------------------------------
    function TopBottomTabCbk(hobj,evdt,indx,numbrow) %#ok<INUSL>
    % Executed when the user clicks on a another tab or resize.

    if numbrow>1 % the user clicked on another row
        for pp=ntab:-1:1, ntab = ntab - strcmp(string(pp),''); end
        if ntab==1, active=1; end

      % Swop rows:
        temp = labels(1);
        labels(1) = labels(numbrow);
        labels(numbrow) = temp;
        clear temp
      % Rebuild the proper list:
        for ii=1:nrows
            temp(ii,1:numel(labels{ii}))=labels{ii};
        end
    else
        active = indx;
        for ii=1:nrows
            temp(ii,1:numel(labels{ii}))=labels{ii};
        end
    end

  % ======================================
    ltpdagui('title',temp,'active',indx);
  % ======================================

    end
    %-------------------------------------------------------------------------

    %-------------------------------------------------------------------------
    function TopBottomTabResizeCbk(varargin)
    % Properly draws the panels.

    units = get(htab,'Units');
    set(htab,'Units','pixels');
    tabpos = get(htab,'Position');
    tabpos(3) = tabpos(3)*0.85;
    set(htab,'Units',units);
```

```
    ntab = numel(string);
    for oo=ntab:-1:1, ntab = ntab - strcmp(string(oo),''); end

    titlepos = [0,0,0,titleheight];
    panelpos = [0,0,tabpos(3),tabpos(4)-titleheight]+margins;
    titlepos(2) = panelpos(2)+panelpos(4);
    panelpos(2) = panelpos(2)-titleheight*(nrows-1)+6;

    n = sum(cellfun('length',string))+2*ntab;
    titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
    upperpos = titlepos(2);
    for k=loop
        titlepos(3) = panelpos(3)*(length(string{k})+2)/n;
        addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
        titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
        titlepos(2) = upperpos - titlepos(4)*(nrows-1)+3+1;
        set(title(k),'Position',titlepos+addtopos);
        titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
    end

  % ======================================================================

    if nrows>1
        for xxx=2:nrows
            localString = labels{xxx};
            ntab = numel(localString);
            for i=ntab:-1:1, ntab = ntab - strcmp(localString(i),''); end
            n = sum(cellfun('length',localString))+2*ntab;
            titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
            titlepos(2) = titlepos(2) + titlepos(4)-2;
            for k=1:ntab
                titlepos(3) = panelpos(3)*(length(localString{k})+2)/n;
                addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
                titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
                if ntab==1, titlepos(3)=titlepos(3)-2; end
                set(titles(xxx,k),'Position',titlepos+addtopos);
                titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
            end
            if ntab<numel(localString)
                for qq=ntab+1:numel(localString)
                    nullpos=[0,0,1,1];
                    set(titles(xxx,qq),'Position',nullpos);
                end
            end
        end
    end

  % ======================================================================

    titlepos = get(title(active),'Position');
    titlepos = titlepos-addtitle.*[active==sign(1),1,active==sign(1),1];
    titlepos(3) = titlepos(3)-addtitle(3)*(active==sign(2));
    ntab = numel(string);
    for i=ntab:-1:1, ntab = ntab - strcmp(string(i),''); end
    if ntab==1, titlepos(3) = titlepos(3)-4; end

    set(title(active),'Position',titlepos,'BackgroundColor',panelbackgroundcolor);
    set(ttext(active),'FontWeight','bold','BackgroundColor',panelbackgroundcolor);

    insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
    set(inset(1),'Position',insetpos);
    try
      insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
      set(inset(2),'Position',insetpos);
      set(inset(3),'Position',insetpos+[2,1-2*sign(4),-4,0]);
    catch
    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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdalib()
%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
%

% N Tateo
% $Id: ltpdalib.m,v 1.1 2008/03/01 13:43:20 nicola Exp $  $Date: 2008/03/01 13:43:20 $

if ~isempty(nargchk(0,0,nargin))
   error('ltpda:ltpdalib:invalidnumberinputs','Too many inputs');
end

% Attempt to open library:
model = 'ltpda_library';

try
   open_system(model);
catch
   error('ltpda:ltpdalib:invalidmodelname',['Could not find LTPDA Library (' model '.mdl).']);
end

% end of ltpdalib.m
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdasim(block)

%             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 $


setup(block);


%% Setup

function setup(block)

  % Register dialog parameter: none, because they're retrieved directly
  % from the memory. This will prevent the user to modify the parameters
  % outside the proper parameters panel:
  block.NumDialogPrms = 0;

  % Register number of input and output ports
  block.NumInputPorts  = 1;
  block.NumOutputPorts = 1;

  % Setup functional port properties to dynamically inherited.
  block.SetPreCompInpPortInfoToDynamic;

  block.InputPort(1).DirectFeedthrough = true;
  block.InputPort(1).DatatypeID = 0;
  block.InputPort(1).Complexity = 0;
  block.OutputPort(1).DatatypeID = 0;
  block.OutputPort(1).Complexity = 0;
  block.SampleTimes = [0 0];
  block.SetAccelRunOnTLC(false);

  block.OutputPort(1).Dimensions = 1;

  % Register methods
  block.RegBlockMethod('SetInputPortSamplingMode',@SetInpPortFrameData);
  block.RegBlockMethod('SetInputPortDimensions',  @SetInpPortDims);
  block.RegBlockMethod('Outputs',                 @Outputs);

  function SetInpPortFrameData(block, idx, fd) %#ok<INUSL>
  block.InputPort(1).SamplingMode = fd;
  block.OutputPort(1).SamplingMode  = fd;

  function SetInpPortDims(block, idx, di)
  block.InputPort(idx).Dimensions = di;


%% Outputs
function Outputs(block)
global LTPDAinvar gl execModels %#ok<NUSED>

% disp(['The name of the currently executed block is ',get_param(get_param(gcbh,'Parent'),'Name')])
% disp(['The current size of LTPDAinvar is ',num2str(size(LTPDAinvar,1))])

currparent   = get_param(gcbh,'Parent');
subsyshandle = get_param(currparent, 'handle');

% Check if it's a partial execution:
if ismember('executionList',evalin('base','who'))
   executionList = evalin('base','executionList');
   blockIndex = -1;
   for i=1:size(executionList,1), if executionList(i)==gcbh, blockIndex = i; break; end; end
   if blockIndex == -1 % this block is not included in the list of those to be executed.
      disp([' . Partial execution: block ',get_param(subsyshandle,'Name'),' will be skipped.'])
      block.OutputPort(1).Data = -1;
      return
   else
    % Check if the block must receive data from Simulink or retrieve them from a previous calculation:
      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(gcbh,'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(gcb, 'tag'));
0141     if strcmp(blocktag,'generic')
0142        paramcommand = get_param(get_param(get_param(gcbh,'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 desidered 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
]);
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
0201       return
0202
0203    catch err
0204        disp(sprintf('Something''s wrong in the calculation of the results of the ''%s''
arithmetic block.',get_param(currparent,'Name')))
0205        disp('The inputs were:')
0206        for i=1:nparams(params)
0207            disp(['Input ',num2str(i),':'])
0208            LTPDAinvar{block.InputPort(1).Data(i),1} %#ok<NOPRT>
0209        end
0210        disp('The equation was:')
0211        disp(eq)
0212        close(findobj('Name','LTPDA Progress Bar'))
0213
0214        rethrow(err)
0215
0216    end
0217   end % for the arithmetic block
0218
0219 % ==========================================================================
0220 %% ==================== RETRIEVE THE PARAMETERS LIST =======================
0221 % ==========================================================================
0222
0223   % Set this block status to 'executed':
0224     set(subsyshandle,'MaskVariables','1');
0225     set(gcbh,'MaskVariables','1');
0226
0227   % The command line to produce the proper parameters list for the block
0228   % currently executed is retrieved from the description of this  parent subsystem
0229     paramcommand = get_param(subsyshandle,'Description');
0230
0231     if ~isempty(paramcommand)
0232         eval(paramcommand);
0233
0234       % Verify if a global variable name has been used:
0235         for i=1:nparams(params) %#ok<NODEF>
0236             if ischar(params.params(i).val)
0237                 try %#ok<ALIGN>
0238                   % params.params(i).val = evalin('base',params.params(i).val);
0239                     params.params(i).val = eval(params.params(i).val);
0240                 catch end
0241             end
0242         end
0243
0244         if ~exist('paramEnabled','var'), paramEnabled = ones(1,nparams(params)); end
0245         currparam = plist();
0246         for jj=1:nparams(params)
0247             if paramEnabled(jj)==1
0248                 try %#ok<ALIGN>
```

```
                    if strcmpi(params.params(jj).key(1:7),'addPar_'), params.params(jj).key(1:7)=[]; end
                    catch end
%                     try % for cell array params:
%                         if isa(params.params(jj).val,'char') && strcmp(params.params(jj).val(1),'{'), params.params(jj).val = eval(params.params(jj).val) ; end
%                     catch
%                     end
                    currparam = append(currparam,params.params(jj));
                end
            end
        else
            currparam = plist();
        end

      % Retrieve parameters from Simulink:
        paramFromSimulink = 0;
        for i=1:nparams(currparam)
             paramKey = currparam.params(i).key;
             paramVal = currparam.params(i).val;
             if (isa(paramVal,'char') && strcmp(paramVal,'-->')) || (isa(paramVal,'cell') && numel(paramVal)==1 && isa(paramVal{1},'char') && strcmp(paramVal{1},'-->'))
                 paramFromSimulink = paramFromSimulink + 1;
                 mux = find_system(currparent,'SearchDepth',1,'LookUnderMasks','all','BlockType','Mux');
                 muxWidths = get_param(mux{1},'CompiledPortWidths');
                 muxWidths = muxWidths.Inport;
                 inputBlock = find_system(currparent,'SearchDepth',1,'LookUnderMasks','all','Name',lower(paramKey));
                 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
                 inputNumb  = str2double(get_param(inputBlock{1},'Tag'));
                 x = 0;
                 for j=1:inputNumb-1
                     x=x+muxWidths(j);
                 end
                 paramVal = [LTPDAinvar{block.InputPort(1).Data(x+1)}];
                 if muxWidths(inputNumb)>1
                     for j=x+2:x+muxWidths(inputNumb)
                         paramVal = [paramVal,LTPDAinvar{block.InputPort(1).Data(j)}];
                     end
                 end
                 currparam = pset(currparam,paramKey,paramVal);
             end
        end

% ========================================================================
%% ======================== CALCULATE THE RESULTS =========================
% ========================================================================

    y = length(block.InputPort(1).Data) - paramFromSimulink;

  % Check to verify the type of function:
    specFunc = {'display'};
    [meth,classes] = strtok(get(get_param(currparent,'Handle'),'Tag'));
    if strcmp(meth,'method')
       clas = strtrim(classes);
       infoObj = eval([clas,'.getInfo(''',blocktag,''')']);
       category = infoObj.mcategory;
    end

    if ismember(blocktag,specFunc)
      % Input, but no parameters and no output
        type = 4;
      % ========================================
    elseif (exist('category','var') && strcmp(category,'Output'))
      % Input and parameters, but no output
        type = 3;
      % ====================================
    elseif ~isempty(utils.prog.find_in_models(currparent,'SearchDepth',1,'LookUnderMasks','all','BlockType',)) || block.InputPort(1).Data(1)==0
      % No input, but parameters and output
        type = 2;
      % ====================================
    else
      % Input, parameters and output
        type = 1;
      % ============================
    end


    switch type
      % ====================================================================
        case 4 % Input, but no parameters and no output
             command = 'feval(blocktag,';
             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
%s function.',blocktag))
0337                disp(sprintf('The name of the currently executed block is
%s.',get_param(currparent,'Name')))
0338                disp(sprintf('The (first) input data ordinal number, in the list of
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
%s function.',blocktag))
0359                disp(sprintf('The name of the currently executed block is
%s.',get_param(currparent,'Name')))
0360                disp(sprintf('The (first) input data ordinal number, in the list of
objects, was %d.',block.InputPort(1).Data(1)))
0361                disp('The plist passed was:')
0362                currparam %#ok<NOPRT>
0363                close(findobj('Name','LTPDA Progress Bar'))
0364                rethrow(err)
0365            end
0366            block.OutputPort(1).Data = 1; % this will be ignored: no output expected.
0367            return;
0368
0369      % ====================================================================
0370        case 2 % No input, but parameters and output
0371            try
0372                outdata = feval(blocktag,currparam);
0373            catch err
0374                disp('2')
0375                disp(sprintf('Something''s wrong in the calculation of the results of the
%s function.',blocktag))
0376                disp(sprintf('The name of the currently executed block is
%s.',get_param(currparent,'Name')))
0377                disp('The plist passed was:')
0378                currparam %#ok<NOPRT>
0379                close(findobj('Name','LTPDA Progress Bar'))
0380                rethrow(err)
0381            end
0382
0383      % ====================================================================
0384        case 1 % Input, parameters and output
0385            try
0386                command = 'outdata = feval(blocktag,';
0387                for i=1:y
0388                   dVal = block.InputPort(1).Data(i);
0389                   if dVal>0, command = [command sprintf('LTPDAinvar{%d},', dVal)]; end
0390                end
0391                command = [command 'currparam);'];
0392                eval(command);
0393            catch err
0394                disp('1')
0395                disp(sprintf('Something''s wrong in the calculation of the results of the
%s function.',blocktag))
0396                disp(sprintf('The name of the currently executed block is
%s.',get_param(currparent,'Name')))
0397                disp(sprintf('The (first) input data ordinal number, in the list of
objects, was %d.',block.InputPort(1).Data(1)))
0398                disp('The plist passed was:')
0399                currparam %#ok<NOPRT>
0400                close(findobj('Name','LTPDA Progress Bar'))
0401                rethrow(err)
0402            end
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),'-
',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 subsytem 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;
end
0490          end
0491          if notUsedAnymore && LTPDAinvar{block.InputPort(1).Data(i),2}==0
0492             LTPDAinvar(block.InputPort(1).Data(i),:) = [];
0493
0494           % Update the execution history:
0495             ltpda_annotation =
find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model');
0496             execHistory = get_param(ltpda_annotation,'UserData');
0497             for xx=1:size(execHistory,1)
0498                for yy=3:size(execHistory,2),
0499                   if execHistory{xx,yy} == block.InputPort(1).Data(i), execHistory{xx,yy}
= -1; end
0500                   if execHistory{xx,yy} > block.InputPort(1).Data(i), execHistory{xx,yy}
= execHistory{xx,yy}-1; end
0501                end
0502                if execHistory{xx,1} == block.InputPort(1).Data(i), execHistory{xx,1} = -
1; end
0503                if execHistory{xx,1} > block.InputPort(1).Data(i), execHistory{xx,1} =
execHistory{xx,1}-1; end
0504             end
0505             set_param(ltpda_annotation,'UserData',execHistory);
0506
0507             disp('*** LTPDA object cleared from memory, being now unused')
0508          end
0509       end
0510    end
0511 end
0512
0513 %==========================================================================
0514
0515
0516     % The output to Simulink is just the ordinal number (ONE AND ONLY ONE
0517     % NUMBER) of the object appended to the global array LTPDAinvar.
0518
0519 % endfunction
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function pan1main(varargin)
% 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 $

global LTPDAinvar LTPDAoutvar currentsystem gl

% Initialize the list of AOs:
InputObjList=''; % OutputObjList='';
UpdateListbox();

scaleX = getappdata(0, 'ltpda_gui_scalefactor');
guiFontSize = round(8*scaleX);

alltimers = timerfindall;
stop(alltimers(2));
set(alltimers(2),'TimerFcn',@ContinuousCheck,'Period',.5);
start(alltimers(2));
set(gcf,'NextPlot','new')
set(gcf,'CloseRequestFcn',{@AskConfirm,'close'})

%%

currPanel   = varargin{1};
if nargin>1, runSim = varargin{2}; else runSim = 0; end
panelDimens = get(currPanel, 'Position');
backColor   = get(currPanel, 'BackgroundColor');

  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');
  uicontrol('Parent',currPanel,'BackgroundColor',[0.9,0.9,0.9],'Units','normalized','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');
  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');

    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');
  currsystem = bdroot;
    if isempty(currsystem) || isempty(find_system(currsystem,'FindAll','on','Type','Annotation','Tag','ltpda model')), currsystem = 'None'; end
  uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Units','normalized','Position',[0.6375 0.915 0.1875 0.040],'String',currsystem,'FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','UserData','currsystem','Style','text');


    AnalysPanPos   = [.0125 .7983 .975 .111]; % in pixels: [10 panelDimens(4)-100 panelDimens(3)-20 55]

  hanalysisPanel = uipanel('Parent',currPanel,'Title','Analyses Control','BackgroundColor',backColor,'FontSize',guiFontSize-1,'Units','normalized','Position',AnalysPanPos);
    logosize = [40 35]*scaleX;
    icon1 = axes('Parent',hanalysisPanel,'Units','normalized','Position',[.01 .05 0.3 0.3]);
    iconSize = getpixelposition(icon1); iconSize(3:4) = logosize; setpixelposition(icon1,round(iconSize));
    image(imread('icongraph.jpg'),'Parent',icon1);
    axis(icon1,'off');
    clear logosize icon1

    nbuttons    = 3;
    buttonWidth = (AnalysPanPos(3)-.0625-.0125*(nbuttons+1))/nbuttons;
    buttonPos   = [AnalysPanPos(1)+.075 .2 buttonWidth .6];
  uicontrol('Parent',hanalysisPanel,'Units','pixels','Units','normalized','Position',buttonPos,'String','New model','TooltipString','Create a new LTPDA Simulink
model','HorizontalAlignment','center','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hNewButtonCallback,'Style','pushbutton');
    buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  uicontrol('Parent',hanalysisPanel,'Units','pixels','Units','normalized','Position',buttonPos,'String','Open model','TooltipString','Open an existing LTPDA Simulink
model','HorizontalAlignment','center','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hOpenCallback,'Style','pushbutton');
    buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  uicontrol('Parent',hanalysisPanel,'Units','pixels','Units','normalized','Position',buttonPos,'String','Save model','TooltipString','Save the currently selected LTPDA Simulink
model','HorizontalAlignment','center','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hSaveCallback,'Style','pushbutton');
```

```
    clear AnalysPanPos nbuttons buttonWidth buttonPos


    DataPanPos   = [.015 .1633 .975 .615]; % in pixels [10 panelDimens(4)-415 panelDimens(3)-20 305];
  hDataPanel   =  uipanel('Parent',currPanel,'Title','Data Control','BackgroundColor',[1,1,1],'FontSize',guiFontSize-1,'Units','normalized','Position',DataPanPos);
    logosize = [48 39]*scaleX;
    icon2 = axes('Parent',hDataPanel,'Units','normalized','Position',[.01 0.81 0.3 0.3]);
    iconSize = getpixelposition(icon2); iconSize(3:4) = logosize; setpixelposition(icon2,round(iconSize));
    image(imread('icondata.jpg'),'Parent',icon2);
    axis(icon2,'off');
    clear logosize icon2


    nbuttons     = 3;
    buttonWidth = (DataPanPos(3)-.0625-.0125*(nbuttons+1))/nbuttons;
    buttonPos    = [DataPanPos(1)+.075 .9 buttonWidth .08];
  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');
%   buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
% uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Create','FontSize',guiFontSize,'Visible','on','Enable','off','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton');
%   buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
% uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Edit','FontSize',guiFontSize,'Visible','on','Enable','off','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton');
    buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  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');
    buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  hExploreButton =  uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Explore','TooltipString','Explore the selected objects with the LTPDA
Explorer','FontSize',guiFontSize,'Visible','on','Enable','on','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton');
    clear nbuttons buttonWidth buttonPos


    nbuttons     = 4;
    buttonWidth = (DataPanPos(3)-.0625-.012510*(nbuttons+1))/nbuttons;
    buttonPos    = [DataPanPos(1)+.075 .8 buttonWidth .08];
% hImportButton =   uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Import','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hImportButtonCallback,'Style','pushbutton');
%   buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Clear all input','TooltipString','Clear from memory the entire list of input
objects','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',{@AskConfirm,'clear'},'Style','pushbutton');
    buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  uicontrol('Parent',hDataPanel,'Units','normalized','HorizontalAlignment','center','Position',buttonPos,'String','Export to workspace','TooltipString','Export the selected objects from LTPDA to MATLAB
workspace','FontSize',guiFontSize,'Visible','on','Enable','on','Callback',@hExportButtonCallback,'Style','pushbutton');
    buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  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');
    buttonPos(1) = buttonPos(1)+buttonWidth+.0125;
  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');
    clear nbuttons buttonWidth buttonPos


    textPos   = [.0125 .67 .5 .083];
  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');
    clear textPos


    nlists    = 2;
    listWidth = (DataPanPos(3)-.0125*(nlists+1))/nlists;
    listPos   = [.0125 .03 listWidth .66];
  InputObjList   =
uicontrol('Parent',hDataPanel,'BackgroundColor',[0.95,0.95,0.95],'HorizontalAlignment','center','Units','normalized','Position',listPos,'String',listInput,'Value',1,'Max',10,'Min',1,'FontSize',guiFontSize,'FontWeight','normal','Visible','on','Enable','on','Callback',@hListObjsCallback,'Tag','InputObjList','Style',
);
    set(hExploreButton,'UserData',InputObjList);
    listPos(1) = listPos(1)+listWidth+.0125;
% Info panel:
  uicontrol('Parent',hDataPanel,'BackgroundColor',[0.94,0.94,1],'Style','listbox','HorizontalAlignment', 'left','Fontsize', guiFontSize,'Units','pixels','Units','normalized','Position',listPos,'Tag','infoPanel');
    hListObjsCallback(InputObjList,1);
    clear DataPanPos nlists listWidth listPos


  hExecutionPanel =  uipanel('Parent',currPanel,'Title','Execution control','BackgroundColor',[1,1,1],'FontSize',guiFontSize-1,'Units','normalized','Position',[.0125 .0222 .975 .121]);
    logosize = [46 40]*scaleX;
    icon3 = axes('Parent',hExecutionPanel,'Units','normalized','Position',[.01 .05 0.3 0.3]);
    iconSize = getpixelposition(icon3); iconSize(3:4) = logosize; setpixelposition(icon3,round(iconSize));
    image(imread('iconsettings.jpg'),'Parent',icon3);
    axis(icon3,'off');
    clear logosize icon3


    if isempty(getappdata(0,'maintainresults')), setappdata(0,'maintainresults',0); end
  uicontrol('Parent',hExecutionPanel,'BackgroundColor',[1 1 1],'HorizontalAlignment','left','Units','normalized','Position',[.1 .166 .3 .5],'Value',getappdata(0,'maintainresults'),'String',' Maintain all results','FontSize',guiFontSize+1,'TooltipString','Enable this if you want to maintain the results
from all blocks at the end of the analysis calculation','ForeGroundColor','black','Visible','on','Enable','off','Callback','setappdata(0,''maintainresults'',get(gco,''Value''))','Style','checkbox','Tag','enableCheckbox');
    if isempty(getappdata(0,'hidenames')), setappdata(0,'hidenames',0); end
  uicontrol('Parent',hExecutionPanel,'BackgroundColor',[1 1 1],'HorizontalAlignment','left','Units','normalized','Position',[.4 0.166 .23 .5],'Value',getappdata(0,'hidenames'),'String',' Hide block names','FontSize',guiFontSize+1,'TooltipString','Enable this to hide blocks names for clarity; this will
work only in the parameters panel','ForeGroundColor','black','Visible','on','Enable','off','Callback','setappdata(0,''hidenames'',get(gco,''Value''))','Style','checkbox','Tag','enableCheckbox');
    if isempty(find_system(bdroot,'SearchDepth',1,'BlockType','SubSystem','Tag','nestedloops')), loopstatus = 0; else loopstatus = 1; end
  uicontrol('Parent',hExecutionPanel,'BackgroundColor',[1 1 1],'HorizontalAlignment','left','Units','normalized','Position',[.7 0.166 .23 .5],'Value',loopstatus,'String',' Enable nested loops','FontSize',guiFontSize+1,'TooltipString','Enable the execution in nested loops, cycling variable values set by
the proper panel','ForeGroundColor','black','Visible','on','Enable','off','Callback',@UpdateNestedLoops,'Style','checkbox','Tag','enableCheckbox');

if runSim == 1, StartExec(); end

%%

% ======================================================================
% ======================================================================
% ============================ FUNCTIONS ===============================
% ======================================================================
% ======================================================================

%%
    %-----------------------------------------------------------------------
    function ContinuousCheck(varargin)
        % This is the function to execute a continuous timer check

      % Keep alive the first timer:
        alltimers = timerfindall;
        if strcmp(get(alltimers(1),'Running'),'off')
            start(alltimers(1))
        end

        currsys = bdroot;
        if ~isempty(currsys) && ~isempty(find_system(currsys,'FindAll','on','Type','Annotation','Tag','ltpda model'))
            set(findobj('UserData', 'currsystem'), 'String', currsys)
            set(findobj('Tag', 'enableCheckbox'), 'enable', 'on')
        else
            set(findobj('UserData', 'currsystem'), 'String', 'None')
            set(findobj('Tag', 'enableCheckbox'), 'enable', 'off')
        end


    end
    %-----------------------------------------------------------------------

%%
    %-----------------------------------------------------------------------
    function LogoutCallback(varargin)
    % Callback function: run when the user click the "Logout" button

        setappdata(0,'ltpda_repo_pass','');
        userID = getappdata(0,'ltpda_repo_user');
        if isempty(userID), userID = ''; end

        [userID passwd] = loginLTPDA(userID,'');
        setappdata(0,'ltpda_repo_user',userID)
        setappdata(0,'ltpda_repo_pass',passwd)

        ltpdagui('Redraw',1);


    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function AskConfirm(varargin)
    % Whenever the user click the Reset Parameters button

      % Ask for confirmation:
        screenSize = get(0,'ScreenSize');
        backColor  = [1 1 1];
        position   = [(screenSize(3)-250)/2,(screenSize(4)-160)/2,250,160];
        confirmFig = figure('Position',position,'Name','Please confirm','Tag','ConfirmReset','Resize','off','NumberTitle','off','Toolbar','none','Menubar','none');
      % Text
        uicontrol('Parent',confirmFig,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[0 position(4)-40 position(3) 20],'String','Are you sure?','FontName','Times New Roman','FontSize',13,'FontWeight','normal','Visible','on','Style','text');
      % Button Yes
        dimension = position;
        logosize = [77 95];
        dimension = [50 , dimension(4)-logosize(2)-50 , logosize];
        yesbut = axes('Parent',confirmFig,'Units','pixels','Position',dimension,'Tag','Yes');
        image(imread('buttonyes2.jpg'),'Parent',yesbut,'ButtonDownFcn','set(findobj(''Tag'',''ConfirmReset''),''UserData'',0); uiresume;','Tag','YesButton');
        axis(yesbut,'off');
```

```
        % Button No
          dimension = position;
          logosize = [77 95];
          dimension = [127 , dimension(4)-logosize(2)-50 , logosize];
          nobut = axes('Parent',confirmFig,'Units','pixels','Position',dimension,'Tag','No');
          image(imread('buttonno2.jpg'),'Parent',nobut,'ButtonDownFcn','set(findobj(''Tag'',''ConfirmReset''),''UserData'',1); uiresume;','Tag','NoButton');
          axis(nobut,'off');

          uiwait;

          if get(confirmFig,'UserData'), delete(confirmFig); return; end

          if strcmp(varargin{3},'close')
             delete(confirmFig);
             disp('Closing LTPDA GUI...');
             delete(timerfindall);
             clear global LTPDAinvar LTPDAoutvar;
             setappdata(0,'ltpda_repo_user',''); setappdata(0,'ltpda_repo_pass','');
             closereq
          elseif strcmp(varargin{3},'clear')
             delete(confirmFig);
             LTPDAinvar={};
             set(InputObjList,'Value',1)
             set(findobj('Tag', 'infoPanel'), 'String', '')
             set_param(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'),'UserData','');
             UpdateListbox();
          end

      end
      %-----------------------------------------------------------------------

%%
      %-----------------------------------------------------------------------
      function hLoadDataButtonCallback(varargin)
      % Callback function: run when the user click the "Add data" button

          filenames = uipickfiles('REFilter','.txt');
          if ~isempty(filenames) && ~isnumeric(filenames(1,1))
              [x,y]=size(filenames);
              for j=1:y
                  ext = filenames{j}(end-2:end);
                  if strcmpi(ext,'mat')
                      varlist = whos('-file',filenames{j});
                      for k=1:numel(varlist)
                          newObj = load(filenames{j},varlist(k).name); %#ok<NASGU>
                          newObj = eval(['newObj.',varlist(k).name]);
                          if isa(newObj,'ltpda_uo')
                              LTPDAinvar = [LTPDAinvar; {newObj,1} ];
                              disp('  + 1 Obj added to the global variable LTPDAinvar')
                          end
                      end
                  else
                      anobject={ao(filenames{j}),1};
                      LTPDAinvar=[LTPDAinvar;anobject];
                      disp('  + 1 AO added to the global variable LTPDAinvar')
                  end
              end
              UpdateListbox();
          end

      end
      %-----------------------------------------------------------------------

      %-----------------------------------------------------------------------
      function hRemoveButtonCallback(varargin)
      % Callback function: run when the user click the "Remove data" button

             index_selected = get(InputObjList,'Value');
             listObjs = get(InputObjList,'String');
             if ~strcmp(listObjs,'The memory is empty')
                 for i=1:numel(index_selected)
                     objSelect = str2double(strtok(listObjs{index_selected(i),1},'.'));
                     LTPDAinvar{objSelect,1} = [];
                 end
                 UpdateListbox();
             end
             set(findobj('Tag', 'infoPanel'), 'String', '')

      end
      %-----------------------------------------------------------------------

      %-----------------------------------------------------------------------
      function hExportButtonCallback(varargin)
      % Callback function: run when the user click the "Export data" button

          currSelObjs   = get(InputObjList,'Value');
          listObjs = get(InputObjList,'String');
          if ~strcmp(listObjs,'The memory is empty')
              selectedObjs  = {};
              for i=1:numel(currSelObjs)
                  objSelect = str2double(strtok(listObjs{currSelObjs(i),1},'.'));
                  selectedObjs = [selectedObjs,LTPDAinvar(objSelect,1)];
              end
              if ~isempty(selectedObjs)
                  for i=1:numel(selectedObjs)
                      objName = [class(selectedObjs{i}),'_',selectedObjs{i}.name];
                      objName = genvarname(objName,evalin('base','who'));
                      assignin('base', objName, selectedObjs{i});
                      disp(['Saved to workspace with name ',objName])
                  end
              end
          end

      end
      %-----------------------------------------------------------------------

      %-----------------------------------------------------------------------
      function hExploreButtonCallback(varargin)
      % Callback function: run when the user click the "Update list" button

          currSelObjs   = get(InputObjList,'Value');
          listObjs = get(InputObjList,'String');
          if ~strcmp(listObjs,'The memory is empty')
              selectedObjs  = {};
              for i=1:numel(currSelObjs)
                  objSelect = str2double(strtok(listObjs{currSelObjs(i),1},'.'));
                  selectedObjs = [selectedObjs,LTPDAinvar(objSelect,1)];
              end

              if ~isempty(selectedObjs)
                  ltpda_explorer(selectedObjs);
              else
                  ltpda_explorer(LTPDAinvar);
              end

              UpdateListbox();
          end

      end
      %-----------------------------------------------------------------------

      %-----------------------------------------------------------------------
      function SaveWorkspaceCallback(varargin)
      % Callback function: run when the user click the "Save all"
      % button

          try
              [FileName,PathName] = uiputfile('*.mat','Save Workspace As');
              if ~isa(FileName,'double'), save(strcat(PathName,FileName),'LTPDAinvar','LTPDAoutvar'); disp('Workspace saved'); end
          catch
          end
      end
      %-----------------------------------------------------------------------

      %-----------------------------------------------------------------------
      function LoadWorkspaceCallback(varargin)
      % Callback function: run when the user click the "Load workspace"
      % button

```

```
        try
            [FileName,PathName] = uigetfile('*.mat','Load Workspace');
            temp = load(strcat(PathName,FileName),'LTPDAinvar');
            LTPDAinvar = [LTPDAinvar ; temp.LTPDAinvar];
            temp = load(strcat(PathName,FileName),'LTPDAoutvar');
            LTPDAoutvar = [LTPDAoutvar ; temp.LTPDAoutvar];
            disp('Workspace loaded')
            clear temp
            UpdateListbox()
        catch
        end
    end
    %--------------------------------------------------------------------------

    %--------------------------------------------------------------------------
    function UpdateNestedLoops(hObject,varargin)
    % Callback function: run when the user click the "Enable nested loops"
    % checkbox.

        if ~isempty(bdroot) && ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
         % it's selected a ltpda simulink analysis diagram.

           status = get(hObject,'Value');
           nestBlock = find_system(bdroot,'SearchDepth',1,'BlockType','SubSystem','Tag','nestedloops');
           try nestBlock = nestBlock{1}; catch end

           if status % The user enabled nested loops. Create the nested loops block:
              if isempty(nestBlock)
                 paramLoop = cell(1,6);
                 nestBlock = add_block('ltpda_library/Commonly Used Blocks/Globals',[bdroot '/Nested loops']);
                 annotation = find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model');
                 position = get_param(annotation,'Position');
                 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 ])
                 set_param(nestBlock,'UserData',paramLoop,'UserDataPersistent','on')
                 ltpdagui('Redraw',8);
              end
           else % The user disabled nested loops. Remove the block:
              if ~isempty(nestBlock), delete_block(nestBlock); end;
              clear paramLoop;
           end;

        else
           disp('*** Warning: the current model is not a valid LTPDA model. Impossible to enable nested loops.');
           set(hObject,'Value',0);
        end
    end
    %--------------------------------------------------------------------------

%%
    %--------------------------------------------------------------------------
    function UpdateListbox(varargin)
    % Updates the listboxes

        %==================================
        % Update the list of active AOs
        xx = size(LTPDAinvar,1);
        listInput=cell(xx,1);
        if xx>0
            for j=1:xx
                listInput{j,1} = num2str(j);
                [rows,columns] = size(LTPDAinvar{j,1});
                if (rows>1 && columns==1) || (rows==1 && columns>1), listInput{j,1} = [listInput{j,1}, ' .(Vector ',num2str(rows),'x',num2str(columns),') '];
                elseif rows>1 && columns>1, listInput{j,1} = [listInput{j,1},  ' .(Matrix ',num2str(rows),'x',num2str(columns),') '];
                end
                switch class(LTPDAinvar{j,1})
                    case 'ao'
                        anobject  = LTPDAinvar{j,1};
                        aoname    = anobject.name;
                        aocreated = char(anobject.created);
                        listInput{j,1} = [listInput{j,1}, ' . AO .' , aoname , '__' , aocreated];
                    case 'plist'
                        paramNumb = nparams(LTPDAinvar{j,1});
                        objcreated = char(LTPDAinvar{j,1}.created);
                        listInput{j,1}=[listInput{j,1}, ' . PLIST .',num2str(paramNumb),' params','__',objcreated];
                    case 'specwin'
                        listInput{j,1}=[listInput{j,1}, ' . SPECWIN .','__','no creation time'];
                    case 'pzmodel'
                        objcreated = char(LTPDAinvar{j,1}.created);
                        listInput{j,1}=[listInput{j,1}, ' . PZMODEL .','__',objcreated];
                    case 'miir'
                        objcreated = char(LTPDAinvar{j,1}.created);
                        listInput{j,1}=[listInput{j,1}, ' . MIIR .','__',objcreated];
                    otherwise
                        if isa(LTPDAinvar{j,1},'ltpda_uo')
                            objclass  = class(LTPDAinvar{j,1});
                            if ~isempty(LTPDAinvar{j,1}), listInput{j,1}=[listInput{j,1}, ' . ',objclass]; end
                        end
                end
            end
            for j=xx:-1:1
                if isempty(listInput{j,1})
                    listInput(j,:)=[];
                end
            end
        else
            listInput{1,1}='The memory is empty';
        end

        InputObjList = findobj('Tag','InputObjList');
        if ~isempty(InputObjList)
            set(InputObjList,'Value',1)
            set(InputObjList,'String',listInput)
        end

    end
    %--------------------------------------------------------------------------

    %--------------------------------------------------------------------------
    function hListObjsCallback(hObject, varargin)
    % This callback is called whenever the user double click upon one of
    % the AOs in use.

        if ~isempty(varargin{1}) && varargin{1}==1 && size(LTPDAinvar,1)>0
                val = get(hObject, 'Value');
                if length(val) == 1
                    objLines = get(hObject, 'String');
                    if isempty(objLines) || isempty(LTPDAinvar{str2double(deblank(strtok(objLines{val}))) ,1})
                        objtxt = '';
                    else
                        objnumb = str2double(deblank(strtok(objLines{val})));
                        objtxt = display(LTPDAinvar{objnumb});
                    end
                else
                    objtxt = '';
                end
                set(findobj('Tag', 'infoPanel'), 'String', objtxt)
        elseif size(LTPDAinvar,1)>0
            if strcmp(get(gcf,'SelectionType'),'normal')
                val = get(hObject, 'Value');
                if length(val) == 1
                    objLines = get(hObject, 'String');
                    objnumb = str2double(deblank(strtok(objLines{val})));
                    try
                        objtxt = display(LTPDAinvar{objnumb});
                    catch
                        objtxt = class(LTPDAinvar{objnumb});
                    end
                    if isempty(LTPDAinvar{objnumb,1}), objtxt=''; end
                else
                    objtxt = '';
                end
                set(findobj('Tag', 'infoPanel'), 'String', objtxt)
            end

            if strcmp(get(gcf,'SelectionType'),'open') && ~isempty(get(hObject, 'String')) && ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
                val = get(hObject, 'Value');
                objLines = get(hObject, 'String');
                number = deblank(strtok(objLines{val}));
                if isempty(LTPDAinvar{str2double(number),1}), return; end
```

```
                set_param(utils.prog.gcbsh,'Selected','off');
                newBlkHandle=add_block('ltpda_library/Input/Object from list', strcat(bdroot,'/AO1'),'MakeNameUnique', 'on');
                children=get_param(newBlkHandle,'Blocks');
                inputblock=strcat(get(newBlkHandle,'Path'),'/',get(newBlkHandle,'Name'),'/',children{1});
                set_param(inputblock,'Value',number );
              % Modify the block annotation:
                objName = LTPDAinvar{str2double(number),1}.name;
                if numel(objName)>3 && strcmpi(objName,'none'), objName = class(LTPDAinvar{str2double(number),1}); end
                if numel(objName)>20, objName = [ '...',LTPDAinvar{str2double(number),1}.name(end-20:end)]; end
                set_param(newBlkHandle,'AttributesFormatString',[number,': ',objName]);
                set_param(newBlkHandle,'Selected','on');
            end
        end

    end
    %-----------------------------------------------------------------------
%%
    %-----------------------------------------------------------------------
    function hNewButtonCallback(varargin)
    % Callback function: run when the user click the "New model" button.
    % It starts also the timer associated to the continuous selection check
    % (this must be changed in order to allow multiple system execution).

        hour           = datestr(now,13); % HH:MM:SS
        hour(3)        = hour(4);hour(4)=hour(5);hour(5)=hour(7);hour(6)=hour(8);
        currentsystem = strcat(getappdata(0,'ltpda_repo_user'),'_',datestr(now,7),datestr(now,5),datestr(now,11),'_',hour(1:6));
        if ~isempty(find_system('Name',currentsystem)), return; end
        simulink
        new_system(currentsystem)
        open_system(currentsystem)

        systemHandle = get_param(gcs,'Handle');
        evalin('base','load ltpdaSimConfigSet.mat;');
        ltpdaSimConfigSet = evalin('base','ltpdaSimConfigSet');
        attachConfigSetCopy(systemHandle, ltpdaSimConfigSet);
        setActiveConfigSet(systemHandle, 'LTPDA');
        detachConfigSet(systemHandle, 'Configuration');
        evalin('base','clear ltpdaSimConfigSet;');

        set(systemHandle,'Location',[200 100 1100 600]);
        set(systemHandle,'Toolbar','OFF');

        caption = strcat(gcs,'/1');
        add_block('built-in/Note',caption,'Position', [100 0 0 400],'UserData',1);
        annotationHandle = find_system(gcs, 'FindAll', 'on', 'type', 'annotation');
        caption1 = [''          Analysis designed by ',getappdata(0,'ltpda_repo_user')];
      % caption1b = strcat('creator: ',char(java.lang.System.getProperties.getProperty('user.name')));
        caption2 = ['created (UTC): ',char(time)];
        caption3 = ['                  IP: ',char(getHostAddress(java.net.InetAddress.getLocalHost))];
        caption4 = ['                  OS: ',computer];
        caption5 = ['MATLAB ver: ',getappdata(0,'matlab_version')];
        caption6 = ['   LTPDA ver: ',getappdata(0,'ltpda_version')];
        caption = strvcat(caption1,caption2,caption3,caption4,caption5,caption6); %#ok<VCAT>
        set_param(annotationHandle, 'HorizontalAlignment','left','Text',caption,'Tag','ltpda model')
        set_param(annotationHandle,'ClickFcn','disp(''You''''re not allowed to change this'')')

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function hSaveCallback(varargin)
    % This callback is called whenever the user click the 'Save' button to
    % store the current model on disk.

        if ~isempty(bdroot) && ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
            inputBlockHandles = get_param(find_system(bdroot,'AncestorBlock','ltpda_library/Input/Object from list'),'handle');
            for i=1:numel(inputBlockHandles)
                children   = get_param(inputBlockHandles{i},'Blocks');
                inputblock = strcat(get(inputBlockHandles{i},'Path'),'/',get(inputBlockHandles{i},'Name'),'/',children{1});
                currIndex(i)  = str2double(get_param(inputblock,'Value'));
                if currIndex(i)~=0
                    objcommand = string(LTPDAinvar{currIndex(i)});
                    set_param(inputBlockHandles{i},'Description',objcommand);
                end
            end

            oldName  = bdroot;
            if ~isempty(oldName) && ~isempty(find_system(oldName,'FindAll','on','Type','Annotation','Tag','ltpda model'))
                try
                    try cd(getappdata(0,'DefaultAnalysesDirectory')); catch end
                    [fileName,filePath] = uiputfile([oldName,'.mdl'],'Choose destination folder');
                    fileName = [filePath,fileName];
                    save_system(oldName,fileName)
                    try cd(getappdata(0,'DefaultDataDirectory')); catch end
                catch
                end
            end
        end

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function hOpenCallback(varargin)
    % This callback is called whenever the user click the 'Open' button, to
    % retrieve a previously saved model from disk.

        try
            try cd(getappdata(0,'DefaultAnalysesDirectory')); catch end
            [fileName,filePath] = uigetfile('.mdl','Choose source file');
            try cd(getappdata(0,'DefaultDataDirectory')); catch end
            fileName = [filePath,fileName];
            open_system(fileName)
            simulink

          % Load the globals into memory:
            globBlock = find_system(bdroot,'BlockType','SubSystem','Tag','globals');
            if ~isempty(globBlock), gl = get_param(globBlock{1},'UserData'); end


          % Update the indexes inside 'Object from list':
            inputBlockHandles = get_param(find_system(gcs,'AncestorBlock','ltpda_library/Input/Object from list'),'handle');
            for i=1:numel(inputBlockHandles)
                objcommand = get_param(inputBlockHandles{i},'Description');
                if ~isempty(objcommand)
                    xx         = size(LTPDAinvar,1);
                    objcommand = ['LTPDAinvar{',num2str(xx+1),',1}=',objcommand,';LTPDAinvar{',num2str(xx+1),',2}=0'];
                    eval(objcommand);

                    % Update the pointer inside the input block and the annotation:
                    children   = get_param(inputBlockHandles{i},'Blocks');
                    inputblock = strcat(get(inputBlockHandles{i},'Path'),'/',get(inputBlockHandles{i},'Name'),'/',children{1});
                    set_param(inputblock,'Value',num2str(xx+1)
                    set_param(inputBlockHandles{i},'AttributesFormatString',num2str(xx+1));
                end
            end

            UpdateListbox()
        catch
        end


    end
    %-----------------------------------------------------------------------
%%
    %-----------------------------------------------------------------------
    function StartExec(varargin)
    % This callback is called whenever the user click the 'Start' button.

        if isempty(bdroot) || isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model')), return; end

        disp('============================')
        disp('   Beginning the analysis')
        disp('============================')
        stop(timerfindall)
          button = findobj('Tag','StartButton');
          set(button,'CData',imread('startred.jpg'));
```

```
0647            drawnow
0648          progressBar();
0649          drawnow
0650          currSystem = bdroot;
0651             allBlocks = find_system(currSystem,'LookUnderMasks','all');
0652             for i=2:numel(allBlocks), set_param(allBlocks{i},'LinkStatus','inactive'); set_param(allBlocks{i},'MaskVariables',''); end
0653          addterms(currSystem)
0654
0655        % Check if it's a partial execution:
0656          if ismember('executionList',evalin('base','who')), partialExecution = 1;
0657           % evalin('base','executionList')
0658          else partialExecution = 0;
0659          end
0660
0661        % Clear the previous execution history:
0662          if ~partialExecution
0663             set_param(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'),'UserData','');
0664          end
0665
0666        % Set all objects present at the startup to be maintained:
0667          if size(LTPDAinvar,1)>0, LTPDAinvar(:,2) = {1}; end
0668
0669        % Load the globals from the current model:
0670          globBlock = find_system(bdroot,'BlockType','SubSystem','Tag','globals');
0671          if ~isempty(globBlock), gl = get_param(globBlock{1},'UserData'); end
0672
0673
0674 % % % ==================================================================
0675 % % % Execution with nested loops
0676 % % % ==================================================================
0677 % % if loopstatus==1
0678 % %
0679 % % %   Example:      n  order    appearance
0680 % % %   paramLoop={    1   1  ...   x-------
0681 % % %                  2   2  ...   .x------
0682 % % %                  3   3  ...   ..x-----
0683 % % %                  4   2  ...   .x------
0684 % % %                  5   3  ...   ..x-----
0685 % % %                  6   1  ...   x-------
0686 % %
0687 % %
0688 % %     branches=[];
0689 % %     [xx,yy]=size(paramLoop);
0690 % %   % xx is the total number of parameters to change in loops.
0691 % %
0692 % %     for ii=1:xx
0693 % %         if ii==xx || paramLoop{ii+1,2}<=paramLoop{ii,2}
0694 % %             branches=[branches,ii];
0695 % %         end
0696 % %     end
0697 % %
0698 % %   % 'branches' contains the ordinal number of those rows which do not have
0699 % %   % a children row, ie. the following rows have higher order. In the
0700 % %   % example, branches = [ 3 , 5 , 6 ];
0701 % %
0702 % %     family={};
0703 % %     numbIter={};
0704 % %     for ii=1:numel(branches)
0705 % %         currOrder=paramLoop{branches(ii),2};
0706 % %         family{ii,1}  = 0;
0707 % %         numbIter{ii,1} = 1;
0708 % %         for jj=branches(ii):-1:1
0709 % %             if paramLoop{jj,2}==currOrder
0710 % %                 family{ii,currOrder+1}  = jj;
0711 % %                 family{ii,1}            = family{ii,1}+1;
0712 % %                 numbIter{ii,currOrder+1} = paramLoop{jj,6};
0713 % %                 numbIter{ii,1}           = numbIter{ii,1}*paramLoop{jj,6};
0714 % %                 currOrder                = paramLoop{jj,2}-1;
0715 % %             end
0716 % %         end
0717 % %     end
0718 % %   % 'family' contains all the parents information for each branch: the
0719 % %   % first column contains how many parents there are. In the example:
0720 % %   % family = {  3    1    2    3
0721 % %   %             3    1    4    5
0722 % %   %             1    6                }
0723 % %   % Similarly, numbIter contains the number of iteration necessary to
0724 % %   % complete each param. The first column contains the total for each
0725 % %   % branch/row.
0726 % %   % numbIter   = {  40   2    4    5
0727 % %   %                36   2    6    3
0728 % %   %                10   10              }
0729 % %
0730 % %     for ii=1:numel(branches)
0731 % %         x = family{ii,1}; % x = how many params to change in this branch
0732 % %         status=ones(1,x); % status shows the current step in the loop;
0733 % %                           % [2,3,5] means eg. that we've executed the 2nd
0734 % %                           % value for the first param, the 3rd for the
0735 % %                           % second param, and the 5th for the third param.
0736 % %
0737 % %         for iter=1:numbIter{ii,1}
0738 % %           % ==================================================================
0739 % %           % Here goes what must be executed
0740 % %           % ==================================================================
0741 % %
0742 % %             if x==1
0743 % %                 disp(sprintf('===== Current status is: ====== %d',status(1)));
0744 % %             elseif x==2
0745 % %                 disp(sprintf('===== Current status is: ====== %d %d',status(1),status(2)));
0746 % %             elseif x==3
0747 % %                 disp(sprintf('===== Current status is: ====== %d %d %d',status(1),status(2),status(3)));
0748 % %             end
0749 % %
0750 % %             for nn=1:numel(status)
0751 % %                 value=paramLoop{family{ii,nn+1},9}{status(nn)};
0752 % %                 paramcommand = BuildCommand(value,family{ii,nn+1});
0753 % %                 set(paramLoop{family{ii,nn+1},3},'Description',paramcommand);
0754 % %             end
0755 % %
0756
0757 try
0758
0759 % ==================================================================
0760           % ==================================================================
0761           % ==================================================================
0762           % ==================================================================
0763                           [ttt,xxx,yyy] = sim(currSystem);  %#ok<NASGU>
0764           % ==================================================================
0765           % ==================================================================
0766           % ==================================================================
0767
0768           % ==================================================================
0769 % ==================================================================
0770 % %                 for jj=x:-1:1
0771 % %                     if status(jj)<numbIter{ii,jj+1} % then we still have to complete iterations on i-th row
0772 % %                         status(jj)=status(jj)+1;
0773 % %                         if jj~=x % if we've just increased a line which is not the last one, reset the lower lines
0774 % %                             for kk=jj+1:x
0775 % %                                 status(kk)=1;
0776 % %                             end
0777 % %                         end
0778 % %                         break
0779 % %                     end
0780 % %                 end
0781 % %         end
0782 % %     end
0783 % %
0784 % % end
0785 % % % ==================================================================
0786 % % % End of execution with nested loops
0787 % % % ==================================================================
0788
0789     close(findobj('Name','LTPDA Progress Bar'))
0790
0791   % Remove final results (intermediate were removed during analysis, as
0792   % soon as they became not useful anymore):
0793     if ~getappdata(0,'maintainresults') % && ~partialExecution
0794         for i=size(LTPDAinvar,1):-1:1
0795             if LTPDAinvar{i,2}==0
```

```
                LTPDAinvar(i,:)=[];
              % Update the execution history:
                ltpda_annotation = find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model');
                execHistory = get_param(ltpda_annotation,'UserData');
                for xx=1:size(execHistory,1)
                   for yy=3:size(execHistory,2),
                        if execHistory{xx,yy} == i, execHistory{xx,yy} = -1; end
                        if execHistory{xx,yy} > i, execHistory{xx,yy} = execHistory{xx,yy}-1; end
                   end
                   if execHistory{xx,1} == i, execHistory{xx,1} = -1; end
                   if execHistory{xx,1} > i, execHistory{xx,1} = execHistory{xx,1}-1; end
                end
                set_param(ltpda_annotation,'UserData',execHistory);

             end
        end
    end


    UpdateListbox();
    button = findobj('Tag','StartButton');
    set(button,'CData',imread('startgreen.jpg'));
    evalin('base','clear executionList');
    disp('============================')
    disp('      Analysis concluded')
    disp('============================')
    start(timerfindall)

% ==================================
catch ME
% ==================================

  % Remove intermediate results:
    keepRes = getappdata(0,'maintainresults');
    if ~keepRes
        for i=size(LTPDAinvar,1):-1:1
            if LTPDAinvar{i,2}==0, LTPDAinvar(i,:)=[]; end
        end
    end

    UpdateListbox();
    close(findobj('Name','LTPDA Progress Bar'))
    button = findobj('Tag','StartButton');
    set(button,'CData',imread('startgreen.jpg'));
    evalin('base','clear executionList');
    start(timerfindall)

    rethrow(ME)
end

    end
    %-----------------------------------------------------------------------

%%

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{i}=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 ^

```
function pan2param(varargin)
% 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 $

global params newparams paramEnabled oldparamEnabled paramSets
global LTPDAinvar selBlocks objName noParamsReq functionName guiFontSize

%%

currPanel   = varargin{1};
panelDimens = get(currPanel, 'Position');
backColor   = get(currPanel, 'BackgroundColor');
selBlocks   = 0;

guiFontSize = getappdata(0, 'ltpda_gui_fontsize');
if isempty(guiFontSize), guiFontSize = 10; setappdata(0, 'ltpda_gui_fontsize',10); end
set(gcf,'NextPlot','new')
alltimers = timerfindall;
  stop(alltimers(1));
  set(alltimers(1),'TimerFcn',@ContinuousParamCheck,'Period',0.3);
  start(alltimers(1));


%%
    %-----------------------------------------------------------------------
    function buildplistFigureParam(varargin)
    %-----------------------------------------------------------------------
  % params = []; paramEnabled = []; functionName = []; paramSets = [];
    panelDimens = get(currPanel, 'Position');
```

```
    lineSpacing = 30 * guiFontSize/10;
    setappdata(0,'ltpda_gui_LineSpacing',lineSpacing);
    if ~isempty(findobj(gcf,'Tag','apply'))
       notredrawn = 0; undoStatus = get(findobj(gcf,'Tag','undo'),'enable');
     % deletes only the dynamical part of the panel:
       for i = 1:numel(objName), try delete(objName{i}); catch end; end
       delete(findobj('Parent',currPanel,'TooltipString','Remove this parameter'))
    else
       notredrawn = 1; undoStatus = 'off';
     % deletes all, to redraw the entire panel:
       delete(findobj(gcf,'Parent',currPanel));
    end
    objName = [];
    close(findobj('Tag','expandedEditField'))
if notredrawn % Executes the STATIC part too only if it's not just an update of the panel

% ==========================================================================
% ==========================================================================
% ========================== Build STATIC panel ============================
% ==========================================================================
% ==========================================================================

% --------------------------------------------------------------------------
% Retrieve the parameters:
% --------------------------------------------------------------------------
  if isempty(params) && isempty(newparams) % the user has just selected another block
     paramcommand = get_param(gcbh,'Description');
     noParamsReq=0;
     if isempty(paramcommand)
        paramcommand = RetrievePlist();
        selBlocks = utils.prog.gcbsh;
        for kk=1:length(selBlocks)
           set_param(selBlocks(kk),'Description',paramcommand);
        end
     end
     eval(paramcommand)
     newparams = params;
     oldparamEnabled = paramEnabled;
  end

  if ~exist('paramEnabled','var') && isempty(find(newparams,'Sets')) && noParamsReq==0
     paramEnabled = zeros(1,nparams(newparams));
     oldparamEnabled = paramEnabled;
  end
  childpath = find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');
  if numel(get_param(childpath,'Tag'))>6 && strcmp(get_param(childpath,'Tag'),'generic')
    % Text:
       uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[40 panelDimens(4)-40 100
14],'FontSize',guiFontSize,'String','Function name:','TooltipString','Function name','Visible','on','Style','text');
    % Edit field:
       uicontrol('Parent',currPanel,'Units','pixels','BackgroundColor',backColor,'HorizontalAlignment','center','Position',[150 panelDimens(4)-40 100
16],'FontSize',guiFontSize,'String', functionName,'TooltipString','Name of the function','Visible','on','Callback',@functionNameSet,'Style','edit');
       panelDimens = get(currPanel, 'Position');
       panelDimens(4) = panelDimens(4)-40;
  else
       childpath = find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');
       setappdata(0,'ltpda_currFunctionName',get_param(childpath,'Tag'));
  end

  y=nparams(newparams);

  if y==0 && ~exist('paramSets','var') && noParamsReq==0
       paramcommand=[paramcommand,'noParamsReq=1;'];
       selBlocks = utils.prog.gcbsh;
       for kk=1:length(selBlocks), set_param(selBlocks(kk),'Description',paramcommand); end
       buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
  end

% --------------------------------------------------------------------------
% Fixed buttons:
% --------------------------------------------------------------------------
% Help Button
  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');
% Reset Parameters Button
  uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[75 10 75
guiFontSize*16/10],'FontAngle','italic','FontSize',guiFontSize,'String','Reset','TooltipString','Reset the parameters for this
block','Visible','on','Callback',@ResetButtonCallback,'Style','pushbutton');
% Keep local result
  probe = get_param(gcbh,'MaskHelp');
  if ~isempty(probe) && isa(probe,'char') && strcmp(probe,'probe'), keepRes = 1; else keepRes = 0; end
  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');
% 'Set name' checkbox:
  setName = get(gcbh,'UserData');
  if isempty(setName), setName = 0; end
  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');
% 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(gcb,'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 % ---------------------------------------------------------------------------
0148 % Buttons to apply or undo:
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=nparams(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   end
0220   currGap = 0;
0221 % The vector 'paramPages' contains how many params are contained into every page.
0222   if numel(paramPages) > 1
0223      set(findobj('Tag','paramsPage'),'Title',['Page number ', num2str(currPage)] );
0224      uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[200 39 100 20],'String','Go to page: ','FontName','Times New
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',
{@buildplistFigureParam,ii+1},'Style','pushbutton');
0227      end
0228   elseif isempty(paramPages)
0229      paramPages = 0;
0230   end
0231   firstParam = 1;
0232   if nargin>1
0233      if varargin{3}>numel(paramPages), varargin{3} = numel(paramPages); end
0234      for ii = 1:varargin{3}-1
0235         firstParam = firstParam + paramPages(ii);
0236      end
0237      lastParam = firstParam + paramPages(varargin{3})-1;
0238   else
0239      lastParam = paramPages(1);
0240   end
0241
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
parameter','Visible','on','Enable','on','UserData',ii,'Callback', @RemParamCallback,'Style','pushbutton');
0252         addedPar = 1;
0253         paramkeys{ii}(1:7) = [];
0254      end
0255      catch end
0256
0257      switch class(paramvals{ii})
0258
0259        % ========================================================================
0260          case {'double'}
0261        % ========================================================================
0262            % Key text:
0263              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');
0264            % Key value:
0265              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');
0266              if ~noParamsReq && ~addedPar
0267                  set(objName{ii,2},'Visible','off');
0268                  set(objName{ii,3},'Style','text','BackgroundColor',backColor,'Position',[50 panelDimens(4)-(4+lineSpacing*(currParam+1+currGap)) 120 20]);
0269              end
0270
0271            % Val text:
0272              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');
0273            % Val value:
0274              data = mat2str(paramvals{ii});
0275              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');
0276              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',[],'Callback',@fromSimulinkCallback,'Style','checkbox');
            % Val button to expand the edit field:
              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 panelDimens(4)+3-lineSpacing*(currParam+1+currGap) 13 13],'Visible','on','Callback',@ExpandEdit,'Style','pushbutton');

            % Enable parameters from Simulink:
              if strcmp(paramvals{ii},'-->')
                  set(objName{ii,4},'Visible','off');
                  set(objName{ii,5},'Visible','off');
                  set(objName{ii,6},'Value',1);
                  set(objName{ii,8},'Visible','off');
              end

        % ==========================================================================
          case {'char'}
        % ==========================================================================
            % Key text:
              objName{ii,2} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[40 panelDimens(4)-
(4+lineSpacing*(currParam+1+currGap)) 70 20],'String','Name:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
            % Key value:
              objName{ii,3} =  uicontrol('Parent',currPanel,'Units','pixels','Position',[120 panelDimens(4)-lineSpacing*(currParam+1+currGap) 60
20],'String',paramkeys{ii},'FontSize',guiFontSize+1,'Visible','on','Enable','off','UserData',ii,'Callback',@editNameCallback,'Style','edit');
              if ~noParamsReq && ~addedPar
                  set(objName{ii,2},'Visible','off');
                  set(objName{ii,3},'Style','text','BackgroundColor',backColor,'Position',[50 panelDimens(4)-(4+lineSpacing*(currParam+1+currGap)) 120 20]);
              end

            % Val text:
              objName{ii,4} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[205 panelDimens(4)-
(4+lineSpacing*(currParam+1+currGap)) 40 20],'String','Value:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
            % Val value:
              objName{ii,5} = uicontrol('Parent',currPanel,'Units','pixels','Position',[250 panelDimens(4)-lineSpacing*(currParam+1+currGap) 200
20],'String',['''',paramvals{ii},''''],'Fontsize',guiFontSize+1,'Visible','on','Enable','off','UserData',ii,'Callback',@editValueCallback,'Style','edit');
              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',[],'Callback',@fromSimulinkCallback,'Style','checkbox');
            % Val button to expand the edit field:
              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 panelDimens(4)+3-lineSpacing*(currParam+1+currGap) 13 13],'Visible','on','Callback',@ExpandEdit,'Style','pushbutton');

            % Browse button:
              if strcmpi(paramkeys{ii},'filename')
                  objName{ii,7} =  uicontrol('Parent',currPanel,'Units','pixels','Position',[560 panelDimens(4)-lineSpacing*(currParam+1+currGap) 90
20],'String','Browse','Visible','on','Enable','off','UserData',objName{ii,5},'Callback',@browseCallback,'Style','pushbutton');
                  set(objName{ii,5},'Position',[250 panelDimens(4)-lineSpacing*(currParam+1+currGap) 300 20]);
                  set(objName{ii,6},'Position',[670 panelDimens(4)-lineSpacing*(currParam+1+currGap) 120 20]);
                  delete(objName{ii,8});
              end

              % Enable parameters from Simulink:
              if strcmp(paramvals{ii},'-->')
                  set(objName{ii,4},'Visible','off');
                  set(objName{ii,5},'Visible','off');
                  set(objName{ii,7},'Visible','off');
                  try set(objName{ii,8},'Visible','off'); catch end
                  set(objName{ii,6},'Value',1);
              end

        % ==========================================================================
          case 'specwin'
        % ==========================================================================
              selectWind      = paramvals{ii}.type;
              listWindows         = ['Kaiser',specwin.getTypes];
              selectWindValue = strmatch(selectWind,listWindows,'exact');
            % Window text:
              objName{ii,2} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[40 panelDimens(4)-lineSpacing*(currParam+1+currGap)-5
60 20],'String','Window:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
            % Window type popup:
              objName{ii,3} =  uicontrol('Parent',currPanel,'Units','pixels','Position',[105 panelDimens(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');
            % Length text:
              objName{ii,4} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[220 panelDimens(4)-lineSpacing*(currParam+1+currGap)-5
60 20],'String','Length:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
            % Length value:
              currLength = num2str(length(newparams.params(ii).val.win));
              objName{ii,5} =  uicontrol('Parent',currPanel,'Units','pixels','Position',[280 panelDimens(4)-lineSpacing*(currParam+1+currGap)-2 70
20],'String',num2str(currLength),'TooltipString','Window Length','Visible','on','Enable','off','UserData',ii,'Callback', @WindlengthCallback,'Style','edit');
              if strcmp(selectWind,'Kaiser')
               % PSL text if Kaiser:
                objName{ii,6} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[365 panelDimens(4)-
lineSpacing*(currParam+1+currGap)-5 60 20],'String','PSL:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
               % PSL value if Kaiser:
                currPSL = newparams.params(ii).val.psll;
                objName{ii,7} =  uicontrol('Parent',currPanel,'Units','pixels','Position',[415 panelDimens(4)-lineSpacing*(currParam+1+currGap)-2 60
20],'String',num2str(currPSL),'TooltipString','Peak Sidelobe','Visible','on','Enable','off','UserData',ii,'Callback',@WindPSLCallback,'Style','edit');
              end
            % Simulink checkbox:
              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 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');

        % ========================================================================
          case 'pzmodel'
        % ========================================================================
            % Simulink checkbox:
              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');

              clear pzObj currPoles poleFreqs poleQs poleFQs currZeros zeroFreqs zeroQs zeroFQs
              pzObj = newparams.params(ii).val;
              currPoles = pzObj.poles;
              currZeros = pzObj.zeros;
            % Load button:
              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');
            % Gain text:
              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');
            % Gain value:
              currGain = pzObj.gain;
              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');
            % Poles text:
              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');
            % Zeros text:
              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');
            % Poles value:
              for kk=1:numel(currPoles)
                  poleFreqs(kk,1) = currPoles(kk).f;
                  poleQs(kk,1)    = currPoles(kk).q;
                  if ~isnan(poleQs(kk,1))
                      poleFQs{kk,1}=strcat(num2str(poleFreqs(kk,1)),',',num2str(poleQs(kk,1)));
                  else
                      poleFQs{kk,1}=num2str(poleFreqs(kk,1));
                  end
              end
              if numel(currPoles)>1 || numel(currZeros)>1
                  currGap=currGap+1;
              end
              if isempty(currPoles)
                  poleFQs = '';
              else
                  poleFQs=char(poleFQs);
              end
              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');
            % Zeros value:
              for kk=1:numel(currZeros)
                  zeroFreqs(kk,1) = currZeros(kk).f;
                  zeroQs(kk,1)    = currZeros(kk).q;
                  if ~isnan(zeroQs(kk,1))
                      zeroFQs{kk,1}=strcat(num2str(zeroFreqs(kk,1)),',',num2str(zeroQs(kk,1)));
                  else
                      zeroFQs{kk,1}=num2str(zeroFreqs(kk,1));
                  end
              end
              if isempty(currZeros)
                  zeroFQs = '';
              else
                  zeroFQs = char(zeroFQs);
              end
              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');

        % ========================================================================
          case 'pz'
        % ========================================================================
            % Simulink checkbox:
              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');

              currPoles = newparams.params(ii).val;
              clear poleFreqs poleQs poleFQs
              for kk=1:numel(currPoles)
                  poleFreqs(kk,1) = currPoles(kk).f;
                  poleQs(kk,1)    = currPoles(kk).q;
                  if isnan(poleQs(kk,1))
                     poleFQs{kk,1}=num2str(poleFreqs(kk,1));
                  else
                     poleFQs{kk,1}=strcat(num2str(poleFreqs(kk,1)),',',num2str(poleQs(kk,1)));
                  end

```

```
            end
            poleFQs=char(poleFQs);
          % Poles text:
            objName{ii,2} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[50 panelDimens(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');
            if numel(currPoles)>1, currGap=currGap+1; end
          % Poles value:
            objName{ii,3} =  uicontrol('Parent',currPanel,'Units','pixels','Position',[250 panelDimens(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');

      % ==========================================================================
        case 'cell'
      % ==========================================================================
            if isa(paramvals{ii}{1},'char') && strcmp(paramvals{ii}{1},'-->')
              % Key value:
                objName{ii,3} =  uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[50 panelDimens(4)-5-lineSpacing*(currParam+1+currGap) 120 20],'String',paramkeys{ii},'Visible','on','Enable','on','UserData',ii,'Callback',@editNameCallback,'Style','text');
              % Text: from Simulink environment:
                objName{ii,4} = uicontrol('TooltipString','Retrieve from Simulink','Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[230 panelDimens(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');
                fromSimulink = 1;
            else
              % Key text:
                objName{ii,2} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[40 panelDimens(4)-6-lineSpacing*(currParam+1+currGap) 70 20],'String','Name:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
              % Key value:
                objName{ii,3} =  uicontrol('Parent',currPanel,'Units','pixels','Position',[120 panelDimens(4)-lineSpacing*(currParam+1+currGap) 60 20],'String',paramkeys{ii},'Visible','on','Enable','off','UserData',ii,'Callback',@editNameCallback,'Style','edit');
                if ~noParamsReq && ~addedPar
                    set(objName{ii,2},'Visible','off');
                    set(objName{ii,3},'Style','text','BackgroundColor',backColor,'Position',[50 panelDimens(4)-5-lineSpacing*(currParam+1+currGap) 120 20]);
                end

              % Val text:
                objName{ii,4} =   uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[205 panelDimens(4)-6-lineSpacing*(currParam+1+currGap) 40 20],'String','Value:','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
              % Val value:
                data = utils.prog.cell2str(paramvals{ii});
                objName{ii,5} = uicontrol('Parent',currPanel,'Units','pixels','Position',[250 panelDimens(4)-lineSpacing*(currParam+1+currGap) 130 20],'String',data,'Visible','on','Enable','on','UserData',ii,'Callback',@editValueCallback,'Style','edit');
                fromSimulink = 0;
            end

          % Simulink checkbox:
            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',fromSimulink,'Visible','on','Tag',num2str(ii),'UserData',paramvals{ii},'Callback',@fromSimulinkCallback,'Style','checkbox');


      % ==========================================================================
        otherwise % Other types of param?
      % ==========================================================================
          % Key value:
            objName{ii,3} =  uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[50 panelDimens(4)-5-lineSpacing*(currParam+1+currGap) 120 20],'String',paramkeys{ii},'FontSize',guiFontSize+1,'Visible','on','Enable','off','UserData',ii,'Callback',@editNameCallback,'Style','text');
          % Text: input not supported. Please construct by proper block.
            objName{ii,4} = uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Position',[310 panelDimens(4)-4-lineSpacing*(currParam+1+currGap) 200 20],'String','Unsupported yet','FontName','Times New Roman','FontSize',guiFontSize+1,'FontWeight','normal','Visible','on','Style','text');
            fromSimulink = 0;

          % Simulink checkbox:
            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',fromSimulink,'Visible','on','Tag',num2str(ii),'UserData',paramvals{ii},'Callback',@fromSimulinkCallback,'Style','checkbox');

  % ==========================================================================
    end % of SWITCH the param type
  % Cycle to activate enabled params:
    if paramEnabled(ii) || noParamsReq==1
        [xx,yy]       = size(objName);
        jj=2;
        while jj<=yy && ~isempty(objName{ii,jj})
            try set(objName{ii,jj},'Enable','on'); set(objName{ii,jj},'ForegroundColor',[0 0 0]); catch end
            jj=jj+1;
        end
    end

end % of cycle among params

    end % of buildplistFigureParam function
    %-------------------------------------------------------------------------

%% ========================= Selection Check ==============================
% ==========================================================================
    %-------------------------------------------------------------------------
    function ContinuousParamCheck(varargin)
  % This is the function to execute a continuous check on the status
```

```
  % of the current selection (system and block).

          currSelection = utils.prog.gcbsh;
        % Clear the figure if nothing's selected
          if isempty(selBlocks) && isempty(currSelection), return; end
          if (~isempty(selBlocks) && isempty(currSelection)) || strcmp(get_param(gcs,'BlockDiagramType'),'library')
              delete(findobj(gcf,'Parent',currPanel))
              selBlocks = [];
              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');
              return
          end

          if ~isempty(bdroot) && ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
        % it's selected a ltpda simulink analysis diagram.

          if ((numel(selBlocks)==1 && selBlocks~=gcbh) || length(selBlocks)~=length(currSelection)) && ~isempty(currSelection)
                for i=1:length(currSelection)
                    set_param(currSelection(i),'LinkStatus','inactive');
                    currentTags{i}=get_param(currSelection(i),'Tag');
                end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                if numel(currSelection)>1 % then the user has selected multiple blocks
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  % Check for consistency: all methods? same class?
                    for i=1:numel(currentTags)
                        if ~strcmp(currentTags{i},currentTags{1})
                            ErrorSelectionInconsistent();
                            selBlocks = currSelection;
                            return;
                        end
                    end
                  % Check for consistency: same method?:
                    for i=1:numel(currSelection)
                       childpath = find_system(currSelection(i),'LookUnderMasks','all','BlockType','M-S-Function');
                       functionname{i}=get_param(childpath,'Tag');
                       if ~strcmp(functionname{i},functionname{1}) % different functions, inconsistent selection
                          ErrorSelectionInconsistent();
                          selBlocks = currSelection;
                          return;
                       end
                    end
                end
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

              delete(findobj(gcf,'Parent',currPanel))
              selBlocks = currSelection;
              params = []; newparams = []; paramEnabled = []; oldparamEnabled = []; functionName = ''; paramSets = [];
              if getappdata(0,'hidenames') && strcmp(get(gcbh,'ShowName'),'on'), for i=1:length(selBlocks), set_param(selBlocks(i),'ShowName','off'); end;
              elseif ~getappdata(0,'hidenames') && strcmp(get(gcbh,'ShowName'),'off'), for i=1:length(selBlocks), set_param(selBlocks(i),'ShowName','on'); end;
              end

            % Get the new functionName:
              childpath    = find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');
              functionName = get_param(childpath,'Tag');
              if isempty(functionName)
                 description = get_param(gcbh,'Description');
                 funcname = findstr('functionName=',description);
                 if funcname
                    i = funcname+14;
                    while ~strcmp(description(i),';'), i=i+1; end
                    functionName = description(funcname+14:i-2);
                 end
              end

              switch strtok(get(gcbh,'Tag'))
                % ========================================================
                  case {'function','freepl','method'}
                      buildplistFigureParam()
                % ========================================================
                  case 'arithmetic'
                      arithmetic();
                % ========================================================
                  case 'input'
                      LoadInputParamCallback();
                % ========================================================
                  case {'mux','demux'}
                      LoadMuxParamCallback();
                % ========================================================
                  case 'from'
                      LoadFromBlockCallback();
                % ========================================================
              end % of switch

         end % end of blocks selection
         end   % end of system selection check

     end
     %-----------------------------------------------------------------------
```

```
%% ======================== Double functions ==============================
%  ========================================================================
    %-------------------------------------------------------------------------
    function editNameCallback(hObject, varargin)
         currParamIndex = get(hObject,'UserData');
         newName  = get(hObject,'String');
         oldName  = newparams.params(currParamIndex).key;
         addedPar = 0;
         try if strcmpi(oldName(1:7),'addPar_'), addedPar = 1; oldName(1:7)=[]; end; catch end

         if strcmp(newparams.params(currParamIndex).val,'-->')
             block2change = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','Name',lower(oldName));
             while ~isempty(find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','Name',lower(newName))),  newName = [newName,'x']; end
             set_param(block2change{1},'Name',lower(newName))
         end

         if addedPar, newName = ['addPar_',newName]; end
         setKey(newparams.params(currParamIndex),newName);
       % To change the drawing command:
         if strcmp(newparams.params(currParamIndex).val,'-->'), UpdateInports(lower(newName),2); end

         set(findobj('Tag','apply'),'Enable','on');
         set(findobj('Tag','undo'),'Enable','on');

    end
    %-------------------------------------------------------------------------
    %-------------------------------------------------------------------------
    function editValueCallback(hObject, varargin)

        expandedEdit = 0;
        if numel(get(hObject,'UserData'))>1
            hObject = get(hObject,'UserData');
            orighObject = hObject(2);
            hObject = hObject(1);
            expandedEdit = 1;
        else
            orighObject = hObject;
        end
        currParamIndex = get(hObject,'UserData');
        newValue = get(hObject,'String');

        if isempty(newValue), set(orighObject,'String',num2str(newparams.params(currParamIndex).val)); return; end
        if strcmp(newValue(1),''''),
            paramType = 'char';
            newValue(1) = []; newValue(end) = [];
            for j = numel(newValue)-1:-1:2
                if strcmp(newValue(j),'''') && (~strcmp(newValue(j-1),'''') && ~strcmp(newValue(j+1),''''))
                    newValue = [newValue(1:j),'''',newValue(j+1:end)];
                end
            end
        elseif ~strcmp(newValue(1),'{'),
            paramType = 'double';
        else
            paramType = 'double';
        end

        switch paramType
            case 'double'
                try
                    newValue = evalin('base',newValue);
                    if isnumeric(newValue)
                        set(orighObject,'String',mat2str(newValue))
                    else
                        newparams = pset(newparams,newparams.params(currParamIndex).key,newValue);
                        paramcommand = BuildParamCommand(newparams);
                        for kk=1:length(selBlocks)
                            set_param(selBlocks(kk),'Description',paramcommand);
                        end
                        AnnotationUpdate(newparams);
                        buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
                        return
                    end
                catch
                    oldValue = newparams.params(currParamIndex).val;
                    if ~isa(oldValue,'char'), mat2str(oldValue); end
                    set(orighObject,'String',oldValue)
                    return
                end
        end

        if expandedEdit, paramEnabled(currParamIndex) = 1; end
        newparams = pset(newparams,newparams.params(currParamIndex).key,newValue);

        set(findobj('Tag','apply'),'Enable','on');
        set(findobj('Tag','undo'),'Enable','on');

    end
    %-------------------------------------------------------------------------
    %-------------------------------------------------------------------------
    function browseCallback(hObject, varargin)

```

```
        [meth,clas] = strtok(get_param(gcbh,'Tag'),' ');
        categ = 'none';
        if strcmp(meth,'method')
           clas = strtrim(clas);
           infoObj = eval([clas,'.getInfo('''',functionName,''')']);
           categ = infoObj.mcategory;
        end

        if strcmp(categ,'Output')
            [fileName,filePath] = uiputfile('*.*','Choose destination file');
        else
            [fileName,filePath] = uigetfile('*.*','Choose source file');
        end

        if fileName==0
            return
        else
            [file , ext] = strtok(fileName,'.');
            if strcmp(ext,'.*'), fileName(end)=[]; fileName=[fileName,'dat']; end
            fileName = [filePath,fileName];
        end
        editFieldHandle = get(hObject,'UserData');
        set(editFieldHandle,'String',fileName);

        newValue = fileName;
        newparams = pset(newparams,'FILENAME',newValue);

        set(findobj('Tag','apply'),'Enable','on');
        set(findobj('Tag','undo'),'Enable','on');

    end
    %-----------------------------------------------------------------------
%% ====================== Specwin functions ============================
%  ======================================================================
    %-----------------------------------------------------------------------
    function WindListCallback(hObject, varargin)
        currParamID   = get(hObject,'UserData');
        currWindValue = get(hObject,'Value');
        listWindows   = ['Kaiser',specwin.getTypes];
        currWindow    = listWindows{currWindValue};
        oldWindObj    = newparams.params(currParamID).val;
        oldWindow     = oldWindObj.type;
        if strcmp(currWindow,oldWindow), return;
        else
            currWindLength = str2double(get(objName{currParamID,5},'String'));
            if strcmp(currWindow,'Kaiser'), windobj = specwin(currWindow,currWindLength,round(oldWindObj.psll));
            else windobj = specwin(currWindow,currWindLength);
            end
        end

        newparams=pset(newparams,newparams.params(currParamID).key,windobj);

        set(findobj('Tag','apply'),'Enable','on');
        set(findobj('Tag','undo'),'Enable','on');

        if strcmp(currWindow,'Kaiser') || strcmp(oldWindow,'Kaiser'), buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData')); end

    end
    %-----------------------------------------------------------------------
    %-----------------------------------------------------------------------
    function WindlengthCallback(hObject,varargin)
    % Callback function: run when the user change the value in the Length
    % editable field for a specwin object

         currParamID    = get(hObject,'UserData');
         selectedWind   = newparams.params(currParamID).val.type;
         currWindLength = str2double(get(hObject,'String'));


         if strcmp(selectedWind,'Kaiser')
             windobj = specwin(selectedWind,currWindLength, newparams.params(currParamID).val.psll);
         else
             windobj = specwin(selectedWind,currWindLength);
         end

         newparams=pset(newparams,newparams.params(currParamID).key,windobj);

         set(findobj('Tag','apply'),'Enable','on');
         set(findobj('Tag','undo'),'Enable','on');

    end
    %-----------------------------------------------------------------------
    %-----------------------------------------------------------------------
    function WindPSLCallback(hObject,varargin)
    % Callback function: run when the user change the value in the PSL
    % editable field for a specwin object

         currParamID    = get(hObject,'UserData');
         selectedWind   = newparams.params(currParamID).val.type;
         currWindLength = numel(newparams.params(currParamID).val.win);
         currPSL        = round(str2double(get(hObject,'String')));
```

```
0769          set(hObject,'String',num2str(currPSL))
0770
0771          windobj    = specwin(selectedWind,currWindLength, currPSL);
0772          newparams  = pset(newparams,newparams.params(currParamID).key,windobj);
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          end
0829
0830          pzObj     = newparams.params(currParamIndex);
0831          pzKey     = pzObj.key;
0832          pzObj     = pzObj.val;
0833          if poleORzero, pzObj     = pzmodel(pzObj.gain,pzList,pzObj.zeros);
0834          else pzObj     = pzmodel(pzObj.gain,pzObj.poles,pzList);
0835          end
0836          newparams    = pset(newparams,pzKey,pzObj);
0837
0838          set(findobj('Tag','apply'),'Enable','on');
0839          set(findobj('Tag','undo'),'Enable','on');
0840
0841         if numel(pzObj.poles)>1 || numel(pzObj.zeros)>1, buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData')); end
0842
0843     end
0844     %-----------------------------------------------------------------------
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
```

```
        newparams = pset(newparams,newparams.params(currParamIndex).key,poleList);

        set(findobj('Tag','apply'),'Enable','on');
        set(findobj('Tag','undo'),'Enable','on');

    end
    %-----------------------------------------------------------------------

%% ========================= Zero functions ============================
%  ===========================================================================
    %-----------------------------------------------------------------------
    function zeroEdit(hObject, varargin)
        currParamIndex = get(hObject,'UserData');
        zeros          = get(hObject,'String');
        zeros          = readInput(zeros,2);
        [xx,yy]  = size(zeros);
        zeroList = pz();
        for kk=1:xx
            if (yy>1 && zeros(kk,2)==0) || yy==1 , zeroList(kk)=pz(zeros(kk,1));
            else zeroList(kk)=pz(zeros(kk,1),zeros(kk,2));
            end
        end
        newparams = pset(newparams,newparams.params(currParamIndex).key,zeroList);

        set(findobj('Tag','apply'),'Enable','on');
        set(findobj('Tag','undo'),'Enable','on');

    end
    %-----------------------------------------------------------------------

%% ====================== Fixed Buttons Callbacks ========================
% ==========================================================================
    %-----------------------------------------------------------------------
    function AddParamCallback(hObject,varargin)

        hparamType = get(hObject,'UserData');
        if isa(hparamType,'char') && strcmp(hparamType,'input')
            key = 'input';
            while ~isempty(find(newparams,key)), key =[key,'x']; end
            newparams = append(newparams, param(key,'-->'));
            UpdateInports(key,1);
            paramcommand = BuildParamCommand(newparams);
            for kk=1:length(selBlocks)
                set_param(selBlocks(kk),'Description',paramcommand);
            end
            arithmetic();

        else
            paramTypes = get(hparamType,'String');
            paramType = paramTypes{get(hparamType,'Value')};

            switch paramType
                case {'double','double/char'}
                    key = 'addPar_name';
                    while ~isempty(find(newparams,key))
                        key =[key,'x'];
                    end
                    newparams = append(newparams, param(key,1));
                case 'specwin'
                    newParam = param('addPar_Win',specwin('Kaiser',1000,200));
                    newparams = append(newparams, newParam);
                case 'pzmodel'
                    pzObj = pzmodel(1,[pz(10,2),pz(100)],pz(1));
                    newParam = param('addPar_pzmodel',pzObj);
                    newparams = append(newparams, newParam);
                case '-->'
                    key = 'addPar_name';
                    while find(newparams,key)
                        key = [key,'x'];
                    end
                    newparams = append(newparams, param(key,'-->'));
                    UpdateInports(key,1)
                case 'pz'
                    newparams = append(newparams, param('addPar_pz',pz(1)));

            end

            if nparams(newparams)==1, paramEnabled=1;
            else paramEnabled=[paramEnabled,1];
            end

            set(findobj('Tag','apply'),'Enable','on');
            set(findobj('Tag','undo'),'Enable','on');

            buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
        end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function RemParamCallback(hObject, varargin)
```

```
0951
0952          currParamIndex = get(hObject,'UserData');
0953
0954          paramKey = newparams.params(currParamIndex).key;
0955          paramVal = newparams.params(currParamIndex).val;
0956          newparams = remove(newparams,paramKey);
0957          if ~strcmp(get_param(gcbh,'Tag'),'arithmetic'), paramEnabled(currParamIndex)=[]; end
0958
0959          if strcmp(paramVal,'-->') % it's a data from Simulink: delete the relative inport block.
0960              UpdateInports(paramKey,-1);
0961          end
0962          delete(hObject)
0963
0964          set(findobj('Tag','apply'),'Enable','on');
0965          set(findobj('Tag','undo'),'Enable','on');
0966
0967          if strcmp(get_param(gcbh,'Tag'),'arithmetic'), arithmetic();
0968          else buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
0969          end
0970      end
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
1010      end
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.
```

```
            if varargin{2}==-1, simKey2remove = [simKey,{paramKey}]; else simKey2remove= simKey; end
            for i=1:numel(simKey2remove)
                try %#ok<ALIGN>
                    block2delete = utils.prog.find_in_models(gcb,'SearchDepth',1,'LookUnderMasks','all','Name',lower(simKey2remove{i}));
                    if numel(block2delete)>1, block2delete(1)=[]; end
                    blockLines   = get_param(block2delete{1},'LineHandles');
                    if (blockLines.Outport(1)~=-1 && ~isempty(blockLines.Outport(1))),   delete_line(blockLines.Outport(1)); end
                    delete_block(block2delete);
                catch end
            end

          % Add ALL inner inport blocks anew:
            for i=1:numel(simKey)
                newBlock = add_block('simulink/Sources/In1', [gcb,'/Inport1'],'MakeNameUnique','on');
                nameAlreadyPresent = utils.prog.find_in_models(gcb,'SearchDepth',1,'LookUnderMasks','all','Name',lower(simKey{i}));
                if ~isempty(nameAlreadyPresent)
                    for j=1:numel(nameAlreadyPresent)
                        if ~strcmp(nameAlreadyPresent{j},gcb),
                            set_param(nameAlreadyPresent{j},'Name',[lower(simKey{i}),'_fun']);
                        end
                    end
                end
                set(newBlock,'Name',lower(simKey{i}))
            end

          % Now update the inner structure of lines and mux.
            muxblock   = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Mux');
            try %#ok<ALIGN>
                muxLines = get_param(muxblock{1},'LineHandles');
                for k=1:numel(muxLines.Inport),  if (muxLines.Inport(k)~=-1 && ~isempty(muxLines.Inport(k))),   delete_line(muxLines.Inport(k)); end
                end
                for k=1:numel(muxLines.Outport), if (muxLines.Outport(k)~=-1 && ~isempty(muxLines.Outport(k))), delete_line(muxLines.Outport(k)); end
                end
            catch end
            delete_block(muxblock);
            inports  = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Inport');
            fakeInp  = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Ground');
            inports  = [fakeInp;inports];
            if numel(inports)>1
                muxblock = add_block('built-in/Mux', [gcb,'/Mux']);
                set(muxblock,'Position',[70 , 10 , 73 , 150])
                set(muxblock,'Inputs',num2str(numel(inports)))
            end
            for k=1:numel(inports)
                inports{k}=get_param(inports{k},'Handle');
                blkLines = get(inports{k},'LineHandles');
                for j=1:numel(blkLines.Outport), if (blkLines.Outport(j)~=-1 && ~isempty(blkLines.Outport(j))), delete_line(blkLines.Outport(j)); end
                end
                set(inports{k},'Position',[10 , 10+30*(k-1) , 30 , 30+30*(k-1)])
                set(inports{k},'Tag',num2str(k))
                if numel(inports)>1
                    add_line(gcb,[get(inports{k},'Name'),'/1'],[get(muxblock,'Name'),'/',num2str(k)]);
                end
            end
            funcBlock = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','M-S-Function');
            funcBlock = get_param(funcBlock,'Name');
            if numel(inports)>1
                add_line(gcb,[get(muxblock,'Name'),'/1'],[funcBlock{1},'/1']);
            else
                add_line(gcb,[get(inports{1},'Name'),'/1'],[funcBlock{1},'/1']);
            end

        end
      % ==================================================================


      % Updating the drawing command for the block Mask:
        fakeInp  = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Ground');
        drawingCom = '';
        if numel(simKey)>0 % then there are params 'from simulink' added
            if numel(fakeInp)>0 || strcmp(get_param(gcbh,'Tag'),'arithmetic'), currInp=1; else currInp=str2double(get(findobj('Tag','numbDataInputs'),'String'))+1; end
            for nn=1:numel(simKey)
                drawingCom = [drawingCom,'port_label(''input'',',num2str(currInp),',''',lower(simKey{1}),''');'];
                simKey(1)  = [];
                currInp    = currInp+1;
            end
        else % then the only inport is the regular one
            funcBlock  = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','M-S-Function');
            funcName   = get_param(funcBlock{1},'Tag');
            if numel(funcName)>6 && strcmp(funcName(1:6),'ltpda_'), funcName(1:6)=[]; end
            drawingCom = strcat('disp(''',lower(funcName),''')');
        end
        set(gcbh,'MaskDisplay',drawingCom)

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function numberDataInput(hObject,varargin)
    % Called whenever the user changes the number of data inputs:

```

```
        prevNumb = get(hObject,'UserData');
        newNumb  = str2double(get(hObject,'String'));
        if isnan(newNumb), set(hObject,'String',num2str(prevNumb)); return; end

        fakeInp  = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Ground');
        if ~isempty(fakeInp) && newNumb>0
           blockLines   = get_param(fakeInp{1},'LineHandles');
           if (blockLines.Outport(1)~=-1 && ~isempty(blockLines.Outport(1))),   delete_line(blockLines.Outport(1)); end
           delete_block(fakeInp{1})
        end

        dataInports  = find_system(gcb,'SearchDepth',1,'LookUnderMasks','all','BlockType','Inport');

        paramFromSimulink = 0;
        for ii=1:nparams(newparams)
           paramvals{ii} = newparams.params(ii).val;
           if isa(paramvals{ii},'char') && numel(paramvals{ii})>2 && strcmp(paramvals{ii},'-->'), paramFromSimulink = paramFromSimulink + 1; end
        end

        % To set the proper port number for parameters inport:
        for ii=1:paramFromSimulink
           set_param(dataInports{end-paramFromSimulink+ii},'Port',num2str(newNumb+11));
        end
        % To avoid considering inports for parameters:
        dataInports(end-paramFromSimulink+1:end) = [];

        % To remove all previous data inports:
        for ii=1:numel(dataInports)
           try %#ok<ALIGN>
              blockLines   = get_param(dataInports{ii},'LineHandles');
              if (blockLines.Outport(1)~=-1 && ~isempty(blockLines.Outport(1))),   delete_line(blockLines.Outport(1)); end
              delete_block(dataInports{ii});
           catch end
        end

        % To add new inports for all data expected:
        if newNumb==0, add_block('simulink/Sources/Ground', [gcb,'/data_input'],'MakeNameUnique','on','Name','data'); end
        for ii=1:newNumb
           newBlock = add_block('simulink/Sources/In1', [gcb,'/Inport1'],'MakeNameUnique','on','Name',['data',num2str(ii)]);
           set_param(newBlock,'Port',num2str(ii));
        end

        UpdateInports('',1)

     end
     %-----------------------------------------------------------------------

     %-----------------------------------------------------------------------
     function EnableCallback(hObject,varargin)
     % Callback function: run when the user click on the checkbox associated
     % to each parameter, to enable its setting.

          currParamID   = get(hObject,'UserData');
          [xx,yy]       = size(objName);
          switch get(hObject,'Value')
              case 1 % enable parameter change
                  jj=2;
                  while jj<=yy && ~isempty(objName{currParamID,jj})
                      try set(objName{currParamID,jj},'Enable','on'); set(objName{currParamID,jj},'ForegroundColor',[0 0 0]); catch end
                      jj=jj+1;
                  end
                  paramEnabled(currParamID)=1;
              case 0 % disable parameter change
                  jj=2;
                  while jj<=yy && ~isempty(objName{currParamID,jj})
                          try %#ok<ALIGN>
                              if strcmp(get(objName{currParamID,jj},'Style'),'checkbox') && strcmp(get(objName{currParamID,jj},'String'),'From Simulink')
                              else
                                  set(objName{currParamID,jj},'Enable','off');
                                  set(objName{currParamID,jj},'ForegroundColor',[0.7 0.7 0.7]);
                              end
                          catch end
                          jj=jj+1;
                  end
                  paramEnabled(currParamID)=0;
          end

         set(findobj('Tag','apply'),'Enable','on');
         set(findobj('Tag','undo'),'Enable','on');

     end
     %-----------------------------------------------------------------------

     %-----------------------------------------------------------------------
     function ExpandEdit(hObject,varargin)
     % Callback function: run when the user click on the checkbox associated
     % to each parameter, to enable its setting.

         origEditField = get(hObject,'UserData');
         currParamIndex = get(origEditField,'UserData');
         paramkey = newparams.params(currParamIndex).key;
```

```
        if numel(paramkey)>6 && strcmpi(paramkey(1:7),'addpar_'), paramkey(1:7)=[]; end
        paramval = get(origEditField,'String');

        editFigPos = [200,200,800,300];

        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'));
      % Edit field title:
        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');
      % Edit field:
        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');
      % Ok button:
        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');
      % Cancel button:
        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');
    end
    %-----------------------------------------------------------------------

%% ==================== General subfunctions ==============================
% ========================================================================
    %-----------------------------------------------------------------------
    function [array]=readInput(inputdata, maxnumb, varargin)
          [xx,yy]=size(inputdata);
        % xx is the number or rows, yy is the number of characters in the longest row
        % array={};
          array=[];
          for i=1:xx
              rowpos=1;
              elemidx(i)=1;
              for j=1:yy+1
                  if (j==yy+1 || inputdata(i,j)==',' || strcmp(inputdata(i,j),' ') ) && (j==1 || (j>1 && (~strcmp(inputdata(i,j-1),' '))))
                      if (j==1 || strcmp(inputdata(i,j-1),',')) && strcmp(inputdata(i,j),',')
                          array(i,elemidx(i))=0;
                          elemidx(i)=elemidx(i)+1;
                          rowpos=j+1;
                      else
                        % To consider numbers, eg [1]:
                        % try array{i,elemidx(i)}=eval(inputdata(i,rowpos:j-1));  catch elemidx(i)=elemidx(i)-1; end
                        % To consider strings, eg '1':
                        % array{i,elemidx(i)}=inputdata(i,rowpos:j-1);
                          array(i,elemidx(i))=eval(inputdata(i,rowpos:j-1));
                          rowpos=j+1;
                          elemidx(i)=elemidx(i)+1;
                      end
                  elseif j<yy+1 && strcmp(inputdata(i,j),' ')
                      rowpos=j+1;
                  end
              if maxnumb~=0
                  if elemidx(i)==maxnumb+1, break; end
              end
              end
              elemidx(i)=elemidx(i)-1;
          end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function paramcommand = BuildParamCommand(paramlist,varargin)
    % Reconstruct the command line necessary to build a given plist

       % Initialize of the paramcommand ojbect:
          paramcommand = strcat('params=',string(paramlist),';');

          if strcmpi(get_param(gcbh,'Tag'),'arithmetic'), return; end

          if exist('paramEnabled','var') && ~isempty(paramEnabled)
              paramcommand = [paramcommand,' paramEnabled=',mat2str(paramEnabled),';'];
          end

          if ~exist('newparams','var') || nparams(paramlist)==0, paramcommand='params=plist(); paramEnabled=[];'; end

          if exist('noParamsReq','var') && noParamsReq==1
              paramcommand = [paramcommand,' noParamsReq=1;'];
          end

          description = get_param(gcbh,'Description');
          funcname = findstr('functionName=',description);
          if funcname
              i = funcname+14;
              while ~strcmp(description(i),';'), i=i+1; end
              funcname = description(funcname+13:i-1);
              paramcommand = [paramcommand,' functionName=',funcname,';'];
          end

          if exist('paramSets','var') && ~isempty(paramSets)
              paramcommand = [paramcommand,' paramSets=',string(paramSets),';'];
```

```
        end

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function paramcommand = RetrievePlist(varargin)
    % Retrieve the name of the function from the block, then asks for
    % parameters.

    try functionname = varargin{1};
    catch
        childpath = find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');
        functionname=get_param(childpath,'Tag');
    end

    if isempty(functionname)
        description = get_param(gcbh,'Description');
        funcname = findstr('functionName=',description);
        if funcname
            i = funcname+14;
            while ~strcmp(description(i),';'), i=i+1; end
            functionname = description(funcname+14:i-2);
        end
    end

    setappdata(0,'ltpda_currFunctionName',functionname);

    try
        switch functionname
            case 'generic'
                paramcommand = 'functionName=''generic'';params=plist();paramEnabled=[];';
                return
            case 'arithmetic'
                paramcommand = 'functionName=''arithmetic'';params=plist(''alpha'',''-->'',''beta'',''-->'');';
                return
            otherwise
                [func,currClass] = strtok(get(gcbh,'Tag'));
                currClass(1)=[];
                infoObj = eval([currClass,'.getInfo(''',functionname,''')']);

                if numel(infoObj.sets) > 1
                   paramSets = plist('sets',infoObj.sets,'currSet',1);
                   paramcommand = ['paramSets=',string(paramSets),';','params=plist();'];
                elseif numel(infoObj.sets) == 1 && nparams(infoObj.plists)
                   paramEnabled = zeros(1,nparams(infoObj.plists));
                   paramcommand = ['params=',string(infoObj.plists),'; paramEnabled=',mat2str(paramEnabled),';'];
                else
                   paramcommand = 'params=plist();';
                end

        end
    catch ME
        disp(sprintf('Error: unable to retrieve the parameters required by the function "%s". Maybe it''s not supported yet?',functionname));
        ErrorFuncNotSupported();
        rethrow(ME)
    end

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function AnnotationUpdate(paramlist,varargin)
    % Update the block annotation (the text below the block) with all the
    % current parameters

         annotation = '';
         numparams  = nparams(paramlist);
         firstpar   = 1;

         if strcmpi(get_param(gcbh,'Tag'),'arithmetic')
             eq = get(findobj('Tag','equationField'),'String');
             if ~isempty(eq)
            if numel(eq)>40, eq(1:end-25) = []; eq = ['...',eq]; end
                 for nn=1:length(selBlocks)
                     set_param(selBlocks(nn),'AttributesFormatString',eq);
                 end
             end
             return
         end

         for kk=1:numparams
             if paramEnabled(kk)
                 paramkey = paramlist.params(kk).key;
                 try %#ok<ALIGN>
                 if strcmpi(paramkey(1:7),'addPar_'), paramkey(1:7)=[]; end
                 catch end
                 paramval = paramlist.params(kk).val;
                 switch class(paramval)
                     case 'specwin'
                         windowName = paramval.type;
```

```
                        windowlength = size(paramval.win,2);
                        if firstpar==1 % it's the first parameter
                            annotation = ['(specwin) ',windowName,',',num2str(windowlength)];
                            firstpar=2;
                        else % it's not the first parameter
                            annotation = [annotation,'\n','(specwin) ',windowName,',',num2str(windowlength)];
                        end
                        if strcmp(windowName,'Kaiser')
                            windowPsll = paramval.psll;
                            annotation = [annotation,',',num2str(windowPsll)];
                        end
                    case 'double'
                        paramval = mat2str(paramval);
                        if numel(paramval)>25, paramval=['...',paramval(end-25:end)]; end
                        if firstpar==1, annotation=[lower(paramkey),' ',paramval]; firstpar=2;
                        else annotation = [annotation,'\n',lower(paramkey),' ',num2str(paramval)];
                        end
                    case 'char'
                        if numel(paramval)>25, paramval=['...',paramval(end-25:end)]; end
                        paramval = strrep(paramval,''''',''''');
                        if firstpar==1, annotation=[lower(paramkey),' ',paramval]; firstpar=2;
                        else annotation = [annotation,'\n',lower(paramkey),' ',paramval];
                        end
                    case 'pzmodel'
                        if firstpar==1, annotation='(pzmodel)'; firstpar=2;
                        else annotation = [annotation,'\n','(pzmodel)'];
                        end
                    case 'cell'
                        if isa(paramval{1},'char') && strcmp(paramval{1},'(-->)'), strParKey = '(-->)';
                        else strParKey = utils.prog.cell2str(paramval);
                        end
                        if firstpar==1
                            annotation = [lower(paramkey),' ',strParKey]; firstpar=2;
                        else annotation = [annotation,'\n',lower(paramkey),' ',strParKey];
                        end
                    case 'pz'
                        for mm=1:numel(paramval)
                            poleFreqs(mm,1) = paramval(mm).f;
                            poleQs(mm,1)    = paramval(mm).q;
                            if ~isnan(poleQs(mm,1))
                                poleFQs{mm,1}=strcat(num2str(poleFreqs(mm,1)),',',num2str(poleQs(mm,1)));
                            else
                                poleFQs{mm,1}=num2str(poleFreqs(mm,1));
                            end
                        end
                        poleFQs=char(poleFQs);
                        if firstpar==1
                            annotation = ['pz ',poleFQs(1,:)]; firstpar=2;
                            for j=2:size(poleFQs,1), annotation = [annotation,'\n','(pz) ',poleFQs(j,:)]; end
                        else
                            annotation = [annotation,'\n','pole ',poleFQs(1,:)];
                            for j=2:size(poleFQs,1), annotation = [annotation,'\n','(pz) ',poleFQs(j,:)]; end
                        end
% % %                     case 'zero'
% % %                         for mm=1:numel(paramval)
% % %                             zeroFreqs(mm,1) = paramval(mm).f;
% % %                             zeroQs(mm,1)    = paramval(mm).q;
% % %                             if ~isnan(zeroQs(mm,1))
% % %                                 zeroFQs{mm,1}=strcat(num2str(zeroFreqs(mm,1)),',',num2str(zeroQs(mm,1)));
% % %                             else
% % %                                 zeroFQs{mm,1}=num2str(zeroFreqs(mm,1));
% % %                             end
% % %                         end
% % %                         zeroFQs=char(zeroFQs);
% % %                         if firstpar==1
% % %                             annotation = ['zero ',zeroFQs(1,:)]; firstpar=2;
% % %                             for j=2:size(zeroFQs,1), annotation = [annotation,'\n','(zero) ',zeroFQs(j,:)]; end
% % %                         else
% % %                             annotation = [annotation,'\n','zero ',zeroFQs(1,:)];
% % %                             for j=2:size(zeroFQs,1), annotation = [annotation,'\n','(zero) ',zeroFQs(j,:)]; end
% % %                         end
                    end
                end
            end
            % Finally set the block annotation:
            for nn=1:length(selBlocks), set_param(selBlocks(nn),'AttributesFormatString',annotation); end
        end
        %-----------------------------------------------------------------------

        %-----------------------------------------------------------------------
        function functionNameSet(hObject, varargin)
        % Setting the name of the function in the 'generic' block.

            functionName2 = get(hObject,'String');
            classes = utils.helper.ltpda_userclasses;
            for i=1:numel(classes)
                if ismethod(eval(classes{i}),functionName2)
                    currSel = utils.prog.gcbsh;
                    for j=1:numel(currSel), set(currSel(j),'Tag',['method ',classes{i}]); end
                    break
                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
1526     end
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,'
help'),'NumberTitle','off','Units','pixels','Position',[200,200,500,300],'Resize','on','ResizeFcn','refresh(gcf)','Color', get(0, 'defaultuicontrolbackgroundcolor'));
1580             % Help text:
```

```
            uicontrol('Parent',helpFigure,'Units','normalized','HorizontalAlignment','left','Position',[0 0 1
1],'FontSize',guiFontSize+1,'String',helptext,'Visible','on','max',100,'Style','edit');
        catch
            disp('Something''s wrong with the creation of the Help figure. Please check ''HelpButtonCallback''')
        end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function ResetButtonCallback(varargin)
    % Whenever the user click the Reset Parameters button

       % Ask for confirmation:
        screenSize = get(0,'ScreenSize');
        backColor   = [1 1 1];
        position   = [(screenSize(3)-250)/2,(screenSize(4)-160)/2,250,160];
        confirmFig = figure('Position',position,'Name','Please confirm reset','Tag','ConfirmReset','Resize','off','NumberTitle','off','Toolbar','none','Menubar','none');
       % Text
        uicontrol('Parent',confirmFig,'BackgroundColor',backColor,'HorizontalAlignment','center','Position',[0 position(4)-40 position(3) 20],'String','Are you sure?
','FontName','Times New Roman','FontSize',13,'FontWeight','normal','Visible','on','Style','text');
       % Button Yes
        dimension = position;
        logosize = [77 95];
        dimension = [50 , dimension(4)-logosize(2)-50 , logosize];
        yesbut = axes('Parent',confirmFig,'Units','pixels','Position',dimension,'Tag','Yes');
        image(imread('buttonyes2.jpg'),'Parent',yesbut,'ButtonDownFcn','set(findobj(''Tag'',''ConfirmReset''),''UserData'',0); uiresume;','Tag','YesButton');
        axis(yesbut,'off');
       % Button No
        dimension = position;
        logosize = [77 95];
        dimension = [127 , dimension(4)-logosize(2)-50 , logosize];
        nobut = axes('Parent',confirmFig,'Units','pixels','Position',dimension,'Tag','No');
        image(imread('buttonno2.jpg'),'Parent',nobut,'ButtonDownFcn','set(findobj(''Tag'',''ConfirmReset''),''UserData'',1); uiresume;','Tag','NoButton');
        axis(nobut,'off');

        uiwait;

        if get(confirmFig,'UserData'), delete(confirmFig); return; end
        delete(confirmFig);

        for i=1:length(selBlocks), set_param(selBlocks(i),'Description',''); set_param(selBlocks(i),'AttributesFormatString',''); end
        params = []; newparams = []; paramEnabled = []; oldparamEnabled = []; functionName = '';
        delete(findobj(gcf,'Parent',currPanel));
%         paramcommand = RetrievePlist();
%
%         if findstr(paramcommand,'functionName=''generic''')
%             for i=1:length(selBlocks), set_param(selBlocks(i),'Name','generic'); end
%         end
%         eval(paramcommand)
%         paramEnabled = zeros(1,nparams(newparams));
%         for kk=1:length(selBlocks)
%             set_param(selBlocks(kk),'Description',paramcommand);
%         end
%         AnnotationUpdate(newparams);
%         buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))
        buildplistFigureParam()

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function KeepResultCallback(hObject,varargin)
    % Whenever the user clicks on the checkbox to maintain local results.

        switch get(hObject,'Value')
            case 0
              % Cycle to restore proper color
                [meth,clas] = strtok(get_param(gcbh,'Tag'),' ');
                infoObj = eval([clas,'.getInfo('',getappdata(0,'ltpda_currFunctionName'),'')']);
                switch infoObj.mcategory
                    case 'Constructor'
                        set_param(gcbh,'Backgroundcolor','green')
                        set_param(gcbh,'MaskHelp','')
                    case 'Output'
                        set_param(gcbh,'Backgroundcolor','lightBlue')
                        set_param(gcbh,'MaskHelp','')
                    otherwise
                        set_param(gcbh,'Backgroundcolor','red')
                        set_param(gcbh,'MaskHelp','')
                end
            case 1
                set_param(gcbh,'Backgroundcolor','magenta')
                set_param(gcbh,'MaskHelp','probe')
        end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function SetNameCallback(hObject,varargin)
    % Whenever the user clicks on the checkbox to maintain local results.

```

```
        setName = get(hObject,'Value');
        set(gcbh,'UserData',setName);
        set(gcbh,'UserDataPersistent','on');

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function ApplyButtonCallback(hObject,varargin)
    % Whenever the user clicks on the 'Apply' button.


       % If it's executed automacally (upon button deletion), check whether
       % there's something to apply or not:
         if strcmp(get(hObject,'enable'),'off'), return; end

       % This is to update the old params with those currently stored into
       % the block:
         temp = paramEnabled;
         paramcommand = get_param(gcbh,'Description');
         eval(paramcommand) % This overwrites also the current 'paramEnabled'
         oldparamEnabled = paramEnabled;
         paramEnabled = temp;

       % Now to update the block with the latest changed newparams:
         paramcommand = BuildParamCommand(newparams);
         for kk=1:length(selBlocks)
            set_param(selBlocks(kk),'Description',paramcommand);
         end
         AnnotationUpdate(newparams);

         set(hObject,'enable','off')

       % Redraw the panel only if the user hasn't changed selection or
       % clicked on a different tab:
         if nargin>2 && varargin{2}, buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData')); end

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function UndoButtonCallback(varargin)
    % Whenever the user clicks on the 'Undo' button.

         temp = paramEnabled;
         paramEnabled = oldparamEnabled;
         oldparamEnabled = temp;

         paramcommand = BuildParamCommand(params);
         for kk=1:length(selBlocks)
            set_param(selBlocks(kk),'Description',paramcommand);
         end
         AnnotationUpdate(params);

       % This to swap params and newparams, so it's possible to undo the undo:
         temp = newparams;
         newparams = params;
         params = temp;
         clear temp;

         buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'))

    end
    %-----------------------------------------------------------------------


%% =========== Input, Mux, Demux, From, Arithmetic block ==================
% ========================================================================
    %-----------------------------------------------------------------------
    function LoadInputParamCallback(varargin)
    % This callback is called whenever the user selects an Input block. The
    % callback draws the panel to set the only parameter associated to this
    % type of block: the ordinal number of AO.

        if length(LTPDAinvar)<1
          % Text: 'Please load at least one AO in memory'
            uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-200)/2 panelDimens(4)-100 200
30],'String','Please load at least one AO in memory','Visible','on','Style','text');
          % Button: 'Add data'
            uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-200)/2 panelDimens(4)-150 200 30],'String','Add
data','FontSize',guiFontSize+1,'Visible','on','Enable','on','Callback',@AddButtonCallback,'Style','pushbutton');
        else
          % Text: 'Obj input:'
            uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','left','Position',[100 panelDimens(4)-60 80 20],'String','Object
input:','Visible','on','Style','text');
          % Edit field:
            children    = get_param(gcb,'Blocks');
            inputblock  = strcat(gcb,'/',children{1});
            currAOinput = get_param(inputblock,'Value');
            uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[190 panelDimens(4)-57 50
20],'String',currAOinput,'Visible','on','Enable','on','Callback',@InputEditCallback,'Tag','inputEditField','Style','edit');
```

```
            % Text: 'Objs list:'
              uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','left','Position',[100 panelDimens(4)-135 50 20],'String','Object list:','Visible','on','Style','text');
            % Popup input objs list:
              xx = size(LTPDAinvar,1);
              listInput = cell(xx,1);
              for j=1:xx
                  switch class(LTPDAinvar{j,1})
                      case 'ao'
                          anobject  = LTPDAinvar{j,1};
                          aoname    = anobject.name;
                          aocreated = char(anobject.created);
                          listInput{j,1} = [num2str(j) , ' . AO .' , aoname , '__' , aocreated];
                      case 'plist'
                          paramNumb = nparams(LTPDAinvar{j,1});
                          objcreated = char(LTPDAinvar{j,1}.created);
                          listInput{j,1}=[num2str(j),' . PLIST .',num2str(paramNumb),' params','__',objcreated];
                      case 'specwin'
                          listInput{j,1}=[num2str(j),' . SPECWIN .','__','no creation time'];
                      case 'pzmodel'
                          objcreated = char(LTPDAinvar{j,1}.created);
                          listInput{j,1}=[num2str(j),' . PZMODEL .','__',objcreated];
                      otherwise
                          if isa(LTPDAinvar{j,1},'ltpda_uo')
                              objclass  = class(LTPDAinvar{j,1});
                              if ~isempty(LTPDAinvar{j,1}), listInput{j,1}=[num2str(j),' . ',objclass]; end
                          end
                  end
              end
              for j=xx:-1:1
                  if isempty(listInput{j,1})
                      listInput(j,:)=[];
                  end
              end
              inputPopup = uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[170 panelDimens(4)-130 500 20],'String',listInput,'Visible','on','Enable','on','Callback',@InputPopupCallback,'Tag','inputPopupField','Style','popup');

            % Set the popup list on the proper line:
              for ii = 1:size(listInput,1)
                  if str2double(deblank(strtok(listInput{ii}))) == str2double(currAOinput), break; end
              end
              set(inputPopup,'Value',ii);
          end
      end
      %-----------------------------------------------------------------------

      %-----------------------------------------------------------------------
      function InputEditCallback(hObject, varargin)
      % This callback is called whenever the user modify the input parameter
      % associated to an Input block

            if ~isempty(varargin{1}), newValue = varargin{1};
            else newValue   = str2double(get(hObject,'String'));
            end
          % Check it's an integer:
            newValue = round(newValue);
          % Check if the value is acceptable, ie:
          % - if it's <length(LTPDAinvar)
            if newValue>length(LTPDAinvar)
                newValue=length(LTPDAinvar);
            end
          % - if it's a LTPDA valid object:
            if ~isa(LTPDAinvar{newValue,1},'ltpda_uo')
                children   = get_param(gcbh,'Blocks');
                inputblock = strcat(strcat(get(gcbh,'Path'),'/',get(gcbh,'Name')),'/',children{1});
                oldValue   = get_param(inputblock,'Value');
                set(findobj('Tag','inputEditField'),'String',oldValue);
                return
            end

            set(hObject,'String',num2str(newValue));

          % Set the popup list on the proper line:
            inputPopup = findobj('Tag','inputPopupField');
            listInput = get(inputPopup,'String');
            for ii = 1:size(listInput,1)
                if str2double(deblank(strtok(listInput{ii}))) == newValue, break; end
            end
            set(inputPopup,'Value',ii);

          % Object name for the annotation:
            objName = LTPDAinvar{newValue,1}.name;
            if numel(objName)>3 && strcmpi(objName,'none'), objName = class(LTPDAinvar{newValue,1}); end
            if numel(objName)>20, objName = ['...',LTPDAinvar{newValue,1}.name(end-20:end)]; end
            objName = [num2str(newValue),': ',objName];

            for i=1:length(selBlocks)
              % Since all input blocks are masks, and we want to change the
              % value deep inside a block INSIDE this mask, we can't use the
              % direct handles but the following:
```

```
            children{i}=get_param(selBlocks(i),'Blocks');
            inputblock=strcat(strcat(get(selBlocks(i),'Path'),'/',get(selBlocks(i),'Name')),'/',children{i}{1});
            set_param(inputblock,'Value',num2str(newValue));
          % Set the annotation:
            set_param(selBlocks(i),'AttributesFormatString',objName);
        end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function InputPopupCallback(hObject, varargin)
    % This callback is called whenever the user clicks on the input list in
    % the parameters panel asociated to the 'Input Obj' block

        val = get(hObject, 'Value');
        objLines = get(hObject, 'String');
        objnumb = str2double(deblank(strtok(objLines{val})));

        InputEditCallback(findobj('Tag','inputEditField'),objnumb);

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function AddButtonCallback(varargin)
    % Callback function: run when the user click the "Add data" button

         filenames = uipickfiles('REFilter','.txt');
         if ~isempty(filenames) && ~isnumeric(filenames(1,1))
             [x,y]=size(filenames);
             for j=1:y
                 anobject={ao(filenames{j})};
                 LTPDAinvar=[LTPDAinvar;anobject];
             end
             disp(sprintf('  + %g AOs added to the global variable LTPDAinvar', y))
             buildplistFigureParam([],[],get(findobj('Tag','paramsPage'),'UserData'));
         end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function LoadMuxParamCallback(varargin)
    % This callback is called whenever the user selects a Mux block (or
    % Demux). The callback opens the panel to set the only parameter
    % associated to this type of block: the number of inputs.
    % The function sets also the dimension of the block accordingly.

        if strcmp(get(gcbh,'Tag'),'mux'),
          % Text: 'Number of inputs:'
            uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','left','Position',[100 panelDimens(4)-60 100 20],'String','Number of inputs:','Visible','on','Style','text');
          % Edit field:
            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');
        elseif strcmp(get(gcbh,'Tag'),'demux'),
          % Text: 'Number of outputs:'
            uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','left','Position',[100 panelDimens(4)-60 100 20],'String','Number of outputs:','Visible','on','Style','text');
          % Edit field:
            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');
        end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function MuxEditCallback(hObject, varargin)
    % This callback is called whenever the user modify the number of inputs
    % associated to a Mux block (or outputs to a Demux)

         newValue   = str2double(get(hObject,'String'));
       % Check it's an integer:
         newValue = round(newValue);
         set(hObject,'String',num2str(newValue));
         for i=1:length(selBlocks)
             newDimens=get(selBlocks(i),'Position');
             newDimens(4)=newDimens(2)+25*(newValue+1);
             if strcmp(get(gcbh,'Tag'),'mux'),
                 set(selBlocks(i),'Inputs',num2str(newValue));
             elseif strcmp(get(gcbh,'Tag'),'demux'),
                 set(selBlocks(i),'Outputs',num2str(newValue));
             end
             set(selBlocks(i),'Position',newDimens);
         end
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function LoadFromBlockCallback(varargin)
    % This callback is called whenever the user selects a From block.

```

```
      % Button: 'Find origin'
        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');
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function FindOriginCallback(varargin)
    % This callback is called whenever click on the 'Find Origin' button,
    % having selected a 'From' block.

        varName = get(gcbh,'GotoTag');
        origin = utils.prog.find_in_models(bdroot,'LookUnderMasks','all','BlockType','Goto','GotoTag',varName);
        if numel(origin)>1
          % Error text: there are multiple origins.
            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');
        elseif numel(origin)==1
          % Text: full path of the origin.
            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');
        else
          % Error text: no origin found.
            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');
        end



    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function arithmetic(varargin)
    % This callback is called whenever the user selects an arithmetic block.

        delete(findobj(gcf,'Parent',currPanel))
        lineSpacing = getappdata(0,'ltpda_gui_LineSpacing');
        paramcommand = get_param(gcbh,'Description');
        noParamsReq=0;
        if isempty(paramcommand)
            paramcommand = RetrievePlist('arithmetic');
            selBlocks = utils.prog.gcbsh;
            for kk=1:length(selBlocks)
                set_param(selBlocks(kk),'Description',paramcommand);
            end
        end
        eval(paramcommand)

        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');
        y = nparams(newparams);
        for i=1:y
            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');
            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');
          % Remove input button
            if i>1
                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');
            end
        end
      % Add input button
        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');

      % Output name:
        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');
        outName = get(gcbh,'UserData');
        if isempty(outName), outName = get(gcbh,'Name'); end
        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',);

      % 'Type in the equation' text
        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');
      % Example edit field
        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');

      % 'output =' text
        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');
      % Main edit field
        eqString = get_param(gcbh,'MaskDescription');
        if numel(eqString)>16 && strcmp(eqString(1:17),'Arithmetic block:'), eqString = ''; end
        uicontrol('Parent',currPanel,'BackgroundColor',[0.9 0.9 0.9],'Units','pixels','Position',[180 panelDimens(4)-300 500
```

```
],'String',eqString,'Max',2,'FontSize',guiFontSize+1,'Visible','on','Enable','on','Callback',@eqEdit,'Tag','equationField','Style','edit');

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function eqEdit(hObject,varargin)

        eq = get(hObject,'String');
        if ~isempty(eq)
            if numel(eq)>40, eq(1:end-25) = []; eq = ['...',eq]; end
            for nn=1:length(selBlocks)
                set_param(selBlocks(nn),'MaskDescription',get(hObject,'String'));
                set_param(selBlocks(nn),'AttributesFormatString',eq);
            end
        else
            eqString = get_param(gcbh,'MaskDescription');
            if numel(eqString)>16 && strcmp(eqString(1:17),'Arithmetic block:'), eqString = ''; end
            set(hObject,'String',eqString);
        end

    end
    %-----------------------------------------------------------------------

%% ============================= Errors ================================
% ======================================================================
    %-----------------------------------------------------------------------
    function ErrorFuncNotSupported(varargin)
    % Recalled whenever the parameters call fails

      % Text: '[...] Maybe it''s not supported yet?'
        uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-300)/2 panelDimens(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');
    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function ErrorSelectionInconsistent(varargin)
    % Recalled whenever the user selects inconsistent multiple blocks
        uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','HorizontalAlignment','center','Position',[(panelDimens(4)-300)/2 panelDimens(4)-100-100 300
30],'String','Error: impossible to set unique parameters for the selected blocks','Visible','on','Style','text');
    end
    %-----------------------------------------------------------------------

% ======================================================================
%%

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function pan4import(varargin)

% ==================================================================
% =============== Panel to import data from the workspace ================
% ==================================================================
%
%  $Id: pan4import.m,v 1.8 2008/08/30 12:27:08 nicola Exp $

global LTPDAinvar

ImportObjList='';
UpdateListbox();

currPanel   = varargin{1};
panelDimens = get(currPanel, 'Position');
backColor   = get(currPanel, 'BackgroundColor');

    DataPanPos   = [20 panelDimens(4)-470 panelDimens(3)-40 450];
  hDataPanel    =  uipanel(...
                        'Title','Data currently in the workspace',...
                        'BackgroundColor',[1,1,1],...
                        'FontSize',7,...
                        'Units','pixels',...
                        'Position',DataPanPos);

    nbuttons    = 3;
    buttonWidth = (DataPanPos(3)-10*(nbuttons+1))/nbuttons;
    buttonPos   = [10 DataPanPos(4)-45 buttonWidth 25];
  hPlotButton   =   uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Plot','FontSize',8,'Visible','on','Callback','','Style','pushbutton','enable','off');
    buttonPos(1) = buttonPos(1)+buttonWidth+10;
  hRefreshButton =   uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Refresh','FontSize',8,'Visible','on','Callback',@UpdateListbox,'Style','pushbutton','enable','on');
    buttonPos(1) = buttonPos(1)+buttonWidth+10;
  hImportButton =   uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Import','FontSize',8,'Visible','on','Callback',@ImportCallback,'Style','pushbutton','enable','on');
%     buttonPos(1) = buttonPos(1)+buttonWidth+10;
%   hExploreButton =  uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Explore','FontSize',8,'Visible','on','UserData',ImportObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton','enable','off');
%     buttonPos(1) = buttonPos(1)+buttonWidth+10;
%   hSaveOutButton =  uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Save','FontSize',8,'Visible','on','Enable','off','Callback',@SaveOutputCallback,'Style','pushbutton');
%     clear nbuttons buttonWidth buttonPos


    textWidth = (DataPanPos(3)-10*(3))/2;
    textPos   = [10 DataPanPos(4)-80 textWidth 25];
  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');
    clear textWidth textPos

    nlists    = 2;
    listWidth = (DataPanPos(3)-10*(nlists+1))/nlists;
    listPos   = [10 DataPanPos(4)-315 listWidth 240];
  ImportObjList   =
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',
);
    listPos(1) = listPos(1)+listWidth+10;
  % Info panel:
  uicontrol('Parent',hDataPanel,'BackgroundColor',[0.94,0.94,1],'Style','text','HorizontalAlignment', 'left','Fontsize', 7,'Units','pixels','Position',listPos,'Tag','infoPanel');
    hImportListCallback(ImportObjList,1)
    clear DataPanPos nlists listWidth listPos


%%
    %-------------------------------------------------------------------
    function UpdateListbox(varargin)
    % Updates the listbox of objects in the current workspace

        listInput = evalin('base','who');

        for i=numel(listInput):-1:1
            obj = evalin('base',listInput{i});
            if ~isa(obj,'ltpda_uo') || isa(obj,'cell'), listInput(i)=[]; end
        end

        if isempty(listInput)
            listInput = 'No LTPDA objects currently in the MATLAB workspace';
            set(findobj('Tag', 'ImportObjList'),'Enable', 'off','Value',[],'String',listInput)
        else
            set(findobj('Tag', 'ImportObjList'),'Enable', 'on','Value',[],'String',listInput)
        end

    end
    %-------------------------------------------------------------------
```

```

%%
    %-----------------------------------------------------------------------
    function hImportListCallback(hObject, varargin)
    % This callback is called whenever the user click upon one of the
    % objects in the workspace

        if ~isempty(varargin{1}) && varargin{1}==1 && ~isa(listInput,'char')
             % This is a call only to update the info panel
                val = get(hObject, 'Value');
                if length(val) == 1
                    objLines = get(hObject, 'String');
                    if isempty(objLines)
                        objtxt = '';
                    else
                        objname = objLines{val};
                        obj = evalin('base',objname);
                        objtxt = display(obj);
                    end
                else
                    objtxt = '';
                end
                set(findobj('Tag', 'infoPanel'), 'String', objtxt)
        elseif ~isa(listInput,'char')
            if strcmp(get(gcf,'SelectionType'),'normal')
                val = get(hObject, 'Value');
                if length(val) == 1
                    objLines = get(hObject, 'String');
                    objname = objLines{val};
                    obj = evalin('base',objname);
                    objtxt = display(obj);
                else
                    objtxt = '';
                end
                set(findobj('Tag', 'infoPanel'), 'String', objtxt)
            end
            if strcmp(get(gcf,'SelectionType'),'open') && ~isempty(get(hObject, 'String'))
                val = get(hObject, 'Value');
                objLines = get(hObject, 'String');
                objname = objLines{val};
                obj = evalin('base',objname);
                try %#ok<ALIGN>
                    figure
                    plot(obj);
                catch
                    close(gcf)
                end
            end
        end

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function ImportCallback(varargin)
    % This callback is called whenever the user click the 'Import' button

        objList = findobj('Tag', 'ImportObjList');
        val     = get(objList, 'Value');
        objLines   = get(objList, 'String');
        for i = 1:numel(val)
            objname = objLines{val(i)};
            obj = evalin('base',objname);
            nobj = numel(obj);
            if nobj>1
                for jj=1:nobj
                    objj = {obj(jj) , 0};
                    LTPDAinvar = [LTPDAinvar;objj];
                end
            else
                obj = {obj , 0};
                LTPDAinvar = [LTPDAinvar;obj];
            end
        end
        set(findobj('Tag', 'ImportObjList'),'Value',[])

    end
    %-----------------------------------------------------------------------

%%
end
```

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 ^

```
function pan5output(varargin)

% ========================================================================
% ===================== 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 $

global LTPDAoutvar

InputObjList=''; OutputObjList='';
UpdateListbox();

currPanel   = varargin{1};
panelDimens = get(currPanel, 'Position');
backColor   = get(currPanel, 'BackgroundColor');

    DataPanPos   = [20 panelDimens(4)-320 panelDimens(3)-40 285];
  hDataPanel    =  uipanel(...
                        'Title','Data Control Panel',...
                        'BackgroundColor',[1,1,1],...
                        'FontSize',7,...
                        'Units','pixels',...
                        'Position',DataPanPos);
    nbuttons    = 5;
    buttonWidth = (DataPanPos(3)-10*(nbuttons+1))/nbuttons;
    buttonPos   = [10 DataPanPos(4)-45 buttonWidth 25];
  hExportButton =   uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Export to workspace','FontSize',8,'Visible','on','Callback',@hExportButtonCallback,'Style','pushbutton','enable','on');
    buttonPos(1) = buttonPos(1)+buttonWidth+10;
  hRemoveButton =   uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Remove','FontSize',8,'Visible','on','Callback',@hRemoveButtonCallback,'Style','pushbutton','enable','off');
    buttonPos(1) = buttonPos(1)+buttonWidth+10;
  hRepoButton    =   uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Send to repository','FontSize',8,'Visible','on','Callback',@hClearButtonCallback,'Style','pushbutton','enable','off');
    buttonPos(1) = buttonPos(1)+buttonWidth+10;
  hExploreButton =  uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Explore','FontSize',8,'Visible','on','UserData',InputObjList,'Callback',@hExploreButtonCallback,'Style','pushbutton','enable','on');
%     buttonPos(1) = buttonPos(1)+buttonWidth+10;
%   hSaveOutButton =  uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Save workspace','FontSize',8,'Visible','on','Enable','on','Callback',@SaveWorkspaceCallback,'Style','pushbutton');
    buttonPos(1) = buttonPos(1)+buttonWidth+10;
  hClearButton  =   uicontrol('Parent',hDataPanel,'Units','pixels','HorizontalAlignment','center','Position',buttonPos,'String','Clear output','FontSize',8,'Visible','on','Enable','on','Callback',@hClearButtonCallback,'Style','pushbutton');
    clear nbuttons buttonWidth buttonPos


    ntexts     = 2;
    textWidth = (DataPanPos(3)-10*(ntexts+1))/ntexts;
    textPos   = [10 DataPanPos(4)-80 textWidth 25];
  % 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');
    textPos(1) = textPos(1)+textWidth+10;
  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');
    clear ntexts textWidth textPos


    nlists     = 2;
    listWidth = (DataPanPos(3)-10*(nlists+1))/nlists;
    listPos   = [10 DataPanPos(4)-275 listWidth 200];
  % Info panel:
  uicontrol('Parent',hDataPanel,'BackgroundColor',[0.94,0.94,1],'Style','listbox','HorizontalAlignment', 'left','Fontsize', 8,'Units','pixels','Position',listPos,'Tag','infoPanel');
    listPos(1) = listPos(1)+listWidth+10;
  OutputObjList   =
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',
);
    hResultListCallback(OutputObjList,1)
    clear DataPanPos nlists listWidth listPos


%%
    %-----------------------------------------------------------------------
    function UpdateListbox(varargin)
    % Updates the LTPDAoutvar listbox
```

```
        %==================================
        % Update the list of results AOs
        xx = size(LTPDAoutvar,1);
        listResults = cell(xx,1);
        if xx>0
            for j=1:xx
                listResults{j,1} = num2str(j);
                [rows,columns] = size(LTPDAoutvar{j,1});
                if (rows>1 && columns==1) || (rows==1 && columns>1), listResults{j,1} = [listResults{j,1}, ' .(Vector ',num2str(rows),'x',num2str(columns),') '];
                elseif rows>1 && columns>1, listResults{j,1} = [listResults{j,1}, ' .(Matrix ',num2str(rows),'x',num2str(columns),') '];
                end
                switch class(LTPDAoutvar{j,1})
                    case 'ao'
                        anobject  = LTPDAoutvar{j,1};
                        aoname    = anobject.name;
                        aocreated = char(anobject.created);
                        listResults{j,1} = [listResults{j,1} , ' . AO .' , aoname , '__' , aocreated];
                    case 'plist'
                        paramNumb = nparams(LTPDAoutvar{j,1});
                        objcreated = get(LTPDAoutvar{j,1}.created,'time_str');
                        listResults{j,1}=[listResults{j,1},' . PLIST .',num2str(paramNumb),' params','__',objcreated];
                    case 'specwin'
                        listResults{j,1}=[listResults{j,1},' . SPECWIN .','__','no creation time'];
                    case 'pzmodel'
                        objcreated = get(LTPDAoutvar{j,1}.created,'time_str');
                        listResults{j,1}=[listResults{j,1},' . PZMODEL .','__',objcreated];
                    case 'miir'
                        objcreated = get(LTPDAoutvar{j,1}.created,'time_str');
                        listResults{j,1}=[listResults{j,1},' . MIIR .','__',objcreated];
                    otherwise
                        if isa(LTPDAoutvar{j,1},'ltpda_uo')
                            objclass  = class(LTPDAoutvar{j,1});
                                if ~isempty(LTPDAoutvar{j,1}), listResults{j,1} = [listResults{j,1},' . ',objclass]; end
                        end
                end
            end
            for j=xx:-1:1
                if isempty(listResults{j,1})
                  listResults(j,:)=[];
                end
            end
        else
            listResults{1,1}='The memory is empty';
        end

        OutputObjList = findobj('Tag','OutputObjList');
        if ~isempty(OutputObjList)
            set(OutputObjList,'Value',1)
            set(OutputObjList,'String',listResults)
        end

        %==================================

    end
    %-----------------------------------------------------------------------

%%
    %-----------------------------------------------------------------------
    function hResultListCallback(hObject, varargin)
    % This callback is called whenever the user double click upon one of
    % the AOs in use.
        if ~isempty(varargin{1}) && varargin{1}==1 && size(LTPDAoutvar,1)>0
                val = get(hObject, 'Value');
                if length(val) == 1
                    objLines = get(hObject, 'String');
                    if isempty(objLines)
                        objtxt = '';
                    else
                        objnumb = str2double(deblank(strtok(objLines{val})));
                        objtxt = display(LTPDAoutvar{objnumb});
                    end
                else
                    objtxt = '';
                end
                set(findobj('Tag', 'infoPanel'), 'String', objtxt)
        elseif size(LTPDAoutvar,1)>0
            if strcmp(get(gcf,'SelectionType'),'normal')
                val = get(hObject, 'Value');
                if length(val) == 1
                    objLines = get(hObject, 'String');
                    objnumb = str2double(deblank(strtok(objLines{val})));
                    objtxt = display(LTPDAoutvar{objnumb});
                else
                    objtxt = '';
                end
                set(findobj('Tag', 'infoPanel'), 'String', objtxt)
            end
            if strcmp(get(gcf,'SelectionType'),'open') && ~isempty(get(hObject, 'String'))
                try %#ok<ALIGN>
                    currIndex = get(hObject,'Value');
                    figure
                    plot(LTPDAoutvar{currIndex,1});
                catch
                    close(gcf)
                end
            end
        end

    end
    %-----------------------------------------------------------------------

%     %-----------------------------------------------------------------------
%     function SaveWorkspaceCallback(varargin)
%     % Callback function: run when the user click the "Clear all data"
%     % button
%
%         try
%             [FileName,PathName,FilterIndex] = uiputfile('*.mat','Save Workspace As');
%             if ~isa(FileName,'double'), save(strcat(PathName,FileName),'LTPDAoutvar'); disp('Workspace saved'); end
%             disp('Output saved')
%         catch
%         end
%     end
%     %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function hClearButtonCallback(varargin)
    % Callback function: run when the user click the "Clear all data"
    % button

        LTPDAoutvar={};
        set(OutputObjList,'Value',1)
        set(findobj('Tag', 'infoPanel'), 'String', '')
        UpdateListbox();

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
```

```
    function hExportButtonCallback(varargin)
    % Callback function: run when the user click the "Remove data" button

          currSelObjs   = get(OutputObjList,'Value');
          listObjs = get(OutputObjList,'String');
          if ~strcmp(listObjs,'The memory is empty')
              selectedObjs  = {};
              for i=1:numel(currSelObjs)
                  objSelect = str2double(strtok(listObjs{currSelObjs(i),1},'.'));
                  selectedObjs = [selectedObjs,LTPDAoutvar(objSelect,1)];
              end
              if ~isempty(selectedObjs)
                    for i=1:numel(selectedObjs)
                        objName = [class(selectedObjs{i}),'_',selectedObjs{i}.name];
                        objName = genvarname(objName,evalin('base','who'));
                        assignin('base', objName, selectedObjs{i});
                        disp(['Saved to workspace with name ',objName])
                    end
              end
          end

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function hExploreButtonCallback(varargin)
    % Callback function: run when the user click the "Update list" button

          currSelObjs   = get(OutputObjList,'Value');
          listObjs = get(OutputObjList,'String');
          if ~strcmp(listObjs,'The memory is empty')
              selectedObjs  = {};
              for i=1:numel(currSelObjs)
                  objSelect = str2double(strtok(listObjs{currSelObjs(i),1},'.'));
                  selectedObjs = [selectedObjs,LTPDAoutvar(objSelect,1)];
              end

              if ~isempty(selectedObjs)
                  ltpda_explorer(selectedObjs);
              else
                  ltpda_explorer(LTPDAoutvar);
              end

              UpdateListbox();
          end

    end
    %-----------------------------------------------------------------------


%%
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function pan6globals(varargin)

% ==========================================================================
% ====================== 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 $

global gl currSys
currPanel    = varargin{1};
panelDimens  = get(currPanel, 'Position');
backColor    = get(currPanel, 'BackgroundColor');
selBlocks   = 0;
enlargedFont = 0;
currSys = '';

alltimers = timerfindall;
stop(alltimers(1));
set(alltimers(1),'TimerFcn',@ContinuousParamCheck,'Period',0.3);
start(alltimers(1));

 %-------------------------------------------------------------------------
 function buildGlobalPanel(varargin)
 %-------------------------------------------------------------------------

  globBlock = find_system(bdroot,'SearchDepth',1,'BlockType','SubSystem','Tag','globals');
  if isempty(globBlock) || (~isempty(varargin) && isnumeric(varargin{3}) && varargin{3}==-1)
      gl = struct();
      updateGlobalBlock()
  else
      gl = get_param(globBlock{1},'UserData');
  end

%   if isempty(gl) || (~isempty(varargin) && isnumeric(varargin{3}) && varargin{3}==-1), gl=struct(); end

  panelDimens = get(currPanel, 'Position');
  delete(findobj(gcf,'Parent',currPanel))
  varnames = fieldnames(gl);
  y        = numel(varnames);

% 'List of global variables' text:
  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');
% 'Global names' text:
  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');
% 'Global values' text:
  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');
```

```

  for i=1:y
%     % 'gl.' text:
%       uicontrol('Parent',currPanel,'BackgroundColor',backColor,'Units','pixels','Position',[panelDimens(3)/2-200-20-15 panelDimens(4)-20-24*(i+2)-5 12 20],'String','gl.','FontSize',enlargedFont+8,'Visible','on','Enable','off','Style','text');
     % Variable name edit field:
       uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDimens(3)/2-200-20 panelDimens(4)-20-24*(i+2) 200 20],'String',['gl.',varnames{i}],'FontSize',enlargedFont+8,'Visible','on','Enable','on','UserData',i,'Callback',@globalName,'Style',);
     % Variable value edit field:
       currVal = gl.(varnames{i});
       if iscell(currVal), currVal = cell2str(currVal);
       else currVal = mat2str(currVal);
       end
       uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDimens(3)/2+20 panelDimens(4)-20-24*(i+2) 200 20],'String',currVal,'FontSize',enlargedFont+8,'Visible','on','Enable','on','UserData',i,'Callback',@globalValue,'Style','edit');
     % Remove input button
       uicontrol('Parent',currPanel,'Units','pixels','Position',[700 panelDimens(4)-20-24*(i+2) 20 20],'String','-','Visible','on','Enable','on','Callback', @RemoveGlobal,'UserData',i,'Style','pushbutton');
  end

% Add input button
  uicontrol('Parent',currPanel,'Units','pixels','Position',[panelDimens(3)/2-10 panelDimens(4)-50-24*(y+2) 20 20],'String','+','TooltipString','Add a global variable','Visible','on','Enable','on','Callback',@AddGlobal,'Style','pushbutton');

% Clear all button:
  uicontrol('Parent',currPanel,'Units','pixels','HorizontalAlignment','center','Position',[panelDimens(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');


 end
 %-----------------------------------------------------------------------

%%
    %-------------------------------------------------------------------------
    function ContinuousParamCheck(varargin)
  % This is the function to execute a continuous check on the status
  % of the current selection (system and block).

       % Clear if nothing's selected
          if isempty(bdroot) || strcmp(get_param(bdroot,'BlockDiagramType'),'library') || isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
               delete(findobj(gcf,'Parent',currPanel))
               currSys = gcs;
               gl = struct();
               return
          else
               if ~strcmp(gcs,currSys)
                    % The selection changed.
                    currSys = gcs;
                    buildGlobalPanel();
               end
          end

%           if ~isempty(find_system(bdroot,'FindAll','on','Type','Annotation','Tag','ltpda model'))
%               % It's selected a ltpda simulink analysis diagram.
%               if ~strcmp(gcs,currSys)
%                    % The selection changed.
%                    currSys = gcs;
%                    buildGlobalPanel();
%               end
%           end

    end
    %-------------------------------------------------------------------------

%%
    %-------------------------------------------------------------------------
    function AddGlobal(varargin)

        newGlob  = 'global';
        ii = 1;
        while isfield(gl,[newGlob,num2str(ii)]), ii = ii+1;  end
        newGlob = [newGlob,num2str(ii)];
        gl.(newGlob) = [];
        updateGlobalBlock()
        buildGlobalPanel()

    end
    %-------------------------------------------------------------------------

    %-------------------------------------------------------------------------
    function RemoveGlobal(hObject,varargin)

        currGlob = get(hObject,'Userdata');
        varnames = fieldnames(gl);
        gl = rmfield(gl,varnames{currGlob});
        updateGlobalBlock()
        buildGlobalPanel()

    end
    %-------------------------------------------------------------------------

    %-------------------------------------------------------------------------
    function globalName(hObject,varargin)

        currGlob = get(hObject,'Userdata');
        varnames = fieldnames(gl);
        newGlobName = get(hObject,'String');
        if numel(newGlobName)<4 || ~strcmp(newGlobName(1:3),'gl.'), newGlobName = ['gl.',strrep(newGlobName,'.','')]; end

        newGlobName = ['gl.',genvarname(newGlobName(4:end))];
        if isempty(get(hObject,'String')) || isfield(gl,newGlobName(4:end)), set(hObject,'String',['gl.',varnames{currGlob}]); return; end

        set(hObject,'String',newGlobName);
        gl = utils.prog.rnfield(gl,varnames{currGlob},newGlobName(4:end));
        updateGlobalBlock()
        buildGlobalPanel()

    end
    %-------------------------------------------------------------------------

    %-------------------------------------------------------------------------
    function globalValue(hObject,varargin)

        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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function pan7partial(varargin)

% ==========================================================================
% ====================== Panel for partial execution =======================
% ==========================================================================
%
%  $Id: pan7partial.m,v 1.3 2008/08/30 12:27:08 nicola Exp $

scaleX = getappdata(0, 'ltpda_gui_scalefactor');
guiFontSize = round(8*scaleX);

alltimers = timerfindall;
stop(alltimers(2));
set(alltimers(2),'TimerFcn',@ContinuousCheck,'Period',1);
start(alltimers(2));

%%

currPanel   = varargin{1};
panelDimens = get(currPanel, 'Position');
backColor   = get(currPanel, 'BackgroundColor');

  % Current system:

uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Units','normalized','Pos
,[0.17 0.89 0.125 0.040],'String','Current system:','FontName','Times New
Roman','FontSize',guiFontSize+4,'FontWeight','normal','Visible','on','Style','text');
     currsystem = bdroot;
     if isempty(currsystem) ||
isempty(find_system(currsystem,'FindAll','on','Type','Annotation','Tag','ltpda model')), currsystem =
'None'; end

uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Units','normalized','Pos
,[0.3 0.89 0.1875 0.040],'String',currsystem,'FontName','Times New
Roman','FontSize',guiFontSize+4,'FontWeight','bold','Visible','on','UserData','currsystem','Style','text');
```

```

   % Current block:

uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Units','normalized','Pos
,[0.5 0.89 0.125 0.040],'String','Current block:','FontName','Times New
Roman','FontSize',guiFontSize+4,'FontWeight','normal','Visible','on','Style','text');

uicontrol('Parent',currPanel,'BackgroundColor',backColor,'HorizontalAlignment','left','Units','normalized','Pos
,[0.62 0.89 0.1875 0.040],'String','','FontName','Times New
Roman','FontSize',guiFontSize+4,'FontWeight','bold','Visible','on','UserData','currblock','Style','text');


   % ======================================================================
     FirstPanPos   = [.0125 .56 .975 .26];
   hFirstPanel = uipanel('Parent',currPanel,'BackgroundColor',backColor,'FontSize',guiFontSize-
1,'Units','normalized','Position',FirstPanPos);
     logosize = [53 55]*scaleX;
     icon1 = axes('Parent',hFirstPanel,'Units','normalized','Position',[.03 .45 0.3 0.3]);
     iconSize = getpixelposition(icon1); panelSize = getpixelposition(hFirstPanel); iconSize(3:4) =
logosize; iconSize(2)= (panelSize(4)-iconSize(4))/2; setpixelposition(icon1,iconSize);
     image(imread('iconpar1.jpg'),'Parent',icon1,'Tag','partial1');
     axis(icon1,'off');
     clear logosize iconSize panelSize

     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.');

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');

   % ======================================================================
     SecondPanPos   = [.0125 .29 .975 .26];
   hSecondPanel = uipanel('Parent',currPanel,'BackgroundColor',backColor,'FontSize',guiFontSize-
1,'Units','normalized','Position',SecondPanPos);
     logosize = [53 55]*scaleX;
     icon2 = axes('Parent',hSecondPanel,'Units','normalized','Position',[.03 .45 0.3 0.3]);
     iconSize = getpixelposition(icon2); panelSize = getpixelposition(hFirstPanel); iconSize(3:4) =
logosize; iconSize(2)= (panelSize(4)-iconSize(4))/2; setpixelposition(icon2,iconSize);
     image(imread('iconpar2.jpg'),'Parent',icon2,'Tag','partial2');
     axis(icon2,'off');
     clear logosize iconSize panelSize

     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.');

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');

   % ======================================================================
     ThirdPanPos   = [.0125 .02 .975 .26];
   hThirdPanel = uipanel('Parent',currPanel,'BackgroundColor',backColor,'FontSize',guiFontSize-
1,'Units','normalized','Position',ThirdPanPos);
     logosize = [53 55]*scaleX;
     icon3 = axes('Parent',hThirdPanel,'Units','normalized','Position',[.03 .45 0.3 0.3]);
     iconSize = getpixelposition(icon3); panelSize = getpixelposition(hFirstPanel); iconSize(3:4) =
logosize; iconSize(2)= (panelSize(4)-iconSize(4))/2; setpixelposition(icon3,iconSize);
     image(imread('iconpar3.jpg'),'Parent',icon3,'Tag','partial3');
     axis(icon3,'off');
     clear logosize iconSize panelSize

     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');

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');


     UpdatePanel(1);



%%
     %-----------------------------------------------------------------------
     function ContinuousCheck(varargin)
         % This is the function to execute a continuous timer check

        % Keep alive the first timer:
          alltimers = timerfindall;
          if strcmp(get(alltimers(1),'Running'),'off')
               start(alltimers(1))
          end

          currsys   = bdroot;
          oldsys    = get(findobj('UserData', 'currsystem'), 'String');
          currselection = utils.prog.gcbsh;
          if numel(currselection)>1, UpdatePanel(1); return; end
          if isempty(currsys) || isempty(currselection), UpdatePanel(1); return; end
          currblock = get_param(currselection,'Name');
          oldblock  = get(findobj('UserData', 'currblock'), 'String');
          if strcmp(currsys,oldsys) && strcmp(currblock,oldblock), return; end

```

```
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
0135     end
0136     %-----------------------------------------------------------------------
0137
0138     %-----------------------------------------------------------------------
0139     function UpdatePanel(varargin)
0140        % varargin{1}= 1 if all disabled (not a valid ltpda model or ltpda
0141        %                                 block)
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
0144        %                lacks the necessary inputs.
0145        %              4 model and block valid, all necessary inputs present.
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(gcbh,'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(gcbh,'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     function Run1(varargin)
```

```
     % Execution from the beginning to the current block

     % To find all parents, up to the beginning of the model:
       parentsBlock = unique(utils.prog.findparent(gcbh));
       parentsBlock = [parentsBlock ; find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function');];

       assignin('base','executionList',parentsBlock);
       ltpdagui('redraw',1.1);

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function Run2(varargin)
     % Execution of the current block only

       assignin('base','executionList',find_system(gcbh,'LookUnderMasks','all','BlockType','M-S-Function') );
       ltpdagui('redraw',1.1);

    end
    %-----------------------------------------------------------------------

    %-----------------------------------------------------------------------
    function Run3(varargin)
     % Execution from the current block to the end

     % To find all children, up to the end of the model:
       childrenBlock = unique(utils.prog.findchildren(gcbh));
       childrenBlock(:,2) = 0;
       while min(childrenBlock(:,2))==0
          for i=1:size(childrenBlock,1)
             if childrenBlock(i,2)==0
                nextGenerationChildren = utils.prog.findchildren(childrenBlock(i,1));
                nextGenerationChildren(:,2) = 0;
                if nextGenerationChildren(1,1)~=0
                   childrenBlock = [childrenBlock ; nextGenerationChildren];
                end
                childrenBlock(i,2) = 1;
             end
          end
          childrenBlock = unique(childrenBlock,'rows');
       end
       childrenBlock(:,2)=[];
       childrenBlock = [gcbh ; childrenBlock];

       assignin('base','executionList',childrenBlock);
       ltpdagui('redraw',1.1);

    end
    %-----------------------------------------------------------------------

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = panXyettocome(varargin)

% ==========================================================================
% ======================== 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 $

currPanel   = varargin{1};
panelDimens = get(currPanel, 'Position');
backColor   = get(currPanel, 'BackgroundColor');

uicontrol(...
    'Parent',currPanel,...
    'BackgroundColor',backColor,...
    'Units','pixels',...
    'HorizontalAlignment','center',...
    'Position',[50, (panelDimens(4)-30)/2, panelDimens(3)-100, 30],...
    'Fontname','Times New Roman',...
    'FontSize',13,...
    'FontAngle','italic',...
    'String','This panel is yet to come, but it''s not intended for release in v1.0',...
    'TooltipString','You ain''t seen anything yet',...
    'Visible','on',...
    'Style','text');

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function progressBar(varargin)
% Progress bar for the LTPDA GUI.
%
%%%
```

```
);

% Press X to stop
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');

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;
```

```
clear Browser;

%
% Define information about Signal Viewers
%
Viewer(1).Library = 'simviewers';
Viewer(1).Name    = 'Simulink';

blkStruct.Viewer = Viewer;
clear Viewer;

%
% Define information about Signal Generators
%
Generator(1).Library = 'simgens';
Generator(1).Name    = 'Simulink';

blkStruct.Generator = Generator;
clear Generator;

% Define information for model updater
%blkStruct.ModelUpdaterMethods.fhDetermineBrokenLinks = @UpdateSimulinkBrokenLinksMappingHelper;
blkStruct.ModelUpdaterMethods.fhUpdateModel = @UpdateSimulinkBlocksHelper;

% End of slblocks


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function out = uipickfiles(varargin)
%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.

% Version: 1.0, 25 April 2006
% Author:  Douglas M. Schwarz
% Email:   dmschwarz=ieee*org, dmschwarz=urgrad*rochester*edu
% Real_email = regexprep(Email,{'=','*'},{'@','.'})
% $Id: uipickfiles.m,v 1.1 2008/03/01 13:43:20 nicola Exp $


% Define properties and set default values.
prop.filterspec = '*';
prop.refilter = '';
prop.prompt = 'Select files';
prop.numfiles = [];
prop.output = 'cell';

% Process inputs and set prop fields.
properties = fieldnames(prop);
arg_index = 1;
while arg_index <= nargin
    arg = varargin{arg_index};
    if ischar(arg)
        prop_index = find(strncmpi(arg,properties,length(arg)));
        if length(prop_index) == 1
            prop.(properties{prop_index}) = varargin{arg_index + 1};
        else
            error('Property ''%s'' does not exist or is ambiguous.',arg)
        end
        arg_index = arg_index + 2;
    elseif isstruct(arg)
        arg_fn = fieldnames(arg);
        for i = 1:length(arg_fn)
            prop_index = find(strncmpi(arg_fn{i},properties,...
                length(arg_fn{i})));
            if length(prop_index) == 1
                prop.(properties{prop_index}) = arg.(arg_fn{i});
            else
                error('Property ''%s'' does not exist or is ambiguous.',...
                    arg_fn{i})
            end
        end
        arg_index = arg_index + 1;
    else
        error(['Properties must be specified by property/value pairs',...
            ' or structures.'])
    end
end

```

```
% Validate FilterSpec property.
if isempty(prop.filterspec)
    prop.filterspec = '*';
end
if ~ischar(prop.filterspec)
    error('FilterSpec property must contain a string.')
end

% Validate REFilter property.
if ~ischar(prop.refilter)
    error('REFilter property must contain a string.')
end

% Validate Prompt property.
if ~ischar(prop.prompt)
    error('Prompt property must contain a string.')
end

% Validate NumFiles property.
if numel(prop.numfiles) > 2 || any(prop.numfiles < 0)
    error('NumFiles must be empty, a scalar or two-element vector.')
end
prop.numfiles = unique(prop.numfiles);
if isequal(prop.numfiles,1)
    numstr = 'Select exactly 1 file.';
elseif length(prop.numfiles) == 1
    numstr = sprintf('Select exactly %d files.',prop.numfiles);
else
    numstr = sprintf('Select %d to %d files.',prop.numfiles);
end

% Validate Output property.
legal_outputs = {'cell','struct','char'};
out_idx = find(strncmpi(prop.output,legal_outputs,length(prop.output)));
if length(out_idx) == 1
    prop.output = legal_outputs{out_idx};
else
    error(['Value of ''Output'' property, ''%s'', is illegal or '...
        'ambiguous.'],prop.output)
end


% Initialize file lists.
[current_dir,f,e] = fileparts(prop.filterspec);
filter = [f,e];
if isempty(current_dir)
    current_dir = pwd;
end
if isempty(filter)
    filter = '*';
end
re_filter = prop.refilter;
full_filter = fullfile(current_dir,filter);
path_cell = path2cell(current_dir);
fdir = filtered_dir(full_filter,re_filter);
filenames = {fdir.name}';
filenames = annotate_file_names(filenames,fdir);

% Initialize some data.
file_picks = {};
full_file_picks = {};
dir_picks = dir(' ');  % Create empty directory structure.
show_full_path = false;
nodupes = true;
history = {current_dir};

% Create figure.
gray = get(0,'DefaultUIControlBackgroundColor');
fig = figure('Position',[0 0 740 445],...
    'Color',gray,...
    'WindowStyle','modal',...
    'Resize','off',...
    'NumberTitle','off',...
    'Name',prop.prompt,...
    'IntegerHandle','off',...
    'CloseRequestFcn',@cancel,...
    'CreateFcn',{@movegui,'center'});

% Create uicontrols.
uicontrol('Style','frame',...
    'Position',[255 260 110 70])
uicontrol('Style','frame',...
    'Position',[275 135 110 100])

navlist = uicontrol('Style','listbox',...
    'Position',[10 10 250 320],...
```

```
    'String',filenames,...
    'Value',[],...
    'BackgroundColor','w',...
    'Callback',@clicknav,...
    'Max',2);
pickslist = uicontrol('Style','listbox',...
    'Position',[380 10 350 320],...
    'String',{},...
    'BackgroundColor','w',...
    'Callback',@clickpicks,...
    'Max',2);

openbut = uicontrol('Style','pushbutton',...
    'Position',[270 300 80 20],...
    'String','Open',...
    'Enable','off',...
    'Callback',@open);
arrow = [2 2 2 2 2 2 2 2 1 2 2 2;...
         2 2 2 2 2 2 2 2 2 0 2 2;...
         2 2 2 2 2 2 2 2 2 2 0 2;...
         0 0 0 0 0 0 0 0 0 0 0 0;...
         2 2 2 2 2 2 2 2 2 2 0 2;...
         2 2 2 2 2 2 2 2 2 0 2 2;...
         2 2 2 2 2 2 2 2 1 2 2 2];
arrow(arrow == 2) = NaN;
arrow_im = NaN*ones(16,76);
arrow_im(6:12,45:56) = arrow/2;
im = repmat(arrow_im,[1 1 3]);
addbut = uicontrol('Style','pushbutton',...
    'Position',[270 270 80 20],...
    'String','Add     ',...
    'Enable','off',...
    'CData',im,...
    'Callback',@add);

removebut = uicontrol('Style','pushbutton',...
    'Position',[290 205 80 20],...
    'String','Remove',...
    'Enable','off',...
    'Callback',@remove);
moveupbut = uicontrol('Style','pushbutton',...
    'Position',[290 175 80 20],...
    'String','Move Up',...
    'Enable','off',...
    'Callback',@moveup);
movedownbut = uicontrol('Style','pushbutton',...
    'Position',[290 145 80 20],...
    'String','Move Down',...
    'Enable','off',...
    'Callback',@movedown);

uicontrol('Position',[10 380 250 16],...
    'Style','text',...
    'String','Current Directory',...
    'HorizontalAlignment','center')
dir_popup = uicontrol('Style','popupmenu',...
    'Position',[10 335 250 20],...
    'BackgroundColor','w',...
    'String',path_cell(end:-1:1),...
    'Value',1,...
    'Callback',@dirpopup);
hist_cm = uicontextmenu;
pathbox = uicontrol('Position',[10 360 250 20],...
    'Style','edit',...
    'BackgroundColor','w',...
    'String',current_dir,...
    'HorizontalAlignment','left',...
    'Callback',@change_path,...
    'UIContextMenu',hist_cm);
hist_menus = [];
hist_cb = @history_cb;
hist_menus = make_history_cm(hist_cb,hist_cm,hist_menus,history);

uicontrol('Position',[10 425 80 16],...
    'Style','text',...
    'String','File Filter',...
    'HorizontalAlignment','left')
uicontrol('Position',[100 425 160 16],...
    'Style','text',...
    'String','Reg. Exp. Filter',...
    'HorizontalAlignment','left')
showallfiles = uicontrol('Position',[270 405 100 20],...
    'Style','checkbox',...
    'String','Show All Files',...
    'Value',0,...
    'HorizontalAlignment','left',...
```

```
    'Callback',@togglefilter);
filter_ed = uicontrol('Position',[10 405 80 20],...
    'Style','edit',...
    'BackgroundColor','w',...
    'String',filter,...
    'HorizontalAlignment','left',...
    'Callback',@setfilspec);
refilter_ed = uicontrol('Position',[100 405 160 20],...
    'Style','edit',...
    'BackgroundColor','w',...
    'String',re_filter,...
    'HorizontalAlignment','left',...
    'Callback',@setrefilter);

viewfullpath = uicontrol('Style','checkbox',...
    'Position',[380 335 230 20],...
    'String','Show full paths',...
    'Value',show_full_path,...
    'HorizontalAlignment','left',...
    'Callback',@showfullpath);
remove_dupes = uicontrol('Style','checkbox',...
    'Position',[380 360 230 20],...
    'String','Remove duplicates (as per full path)',...
    'Value',nodupes,...
    'HorizontalAlignment','left',...
    'Callback',@removedupes);
uicontrol('Position',[380 405 350 20],...
    'Style','text',...
    'String','Selected Files',...
    'HorizontalAlignment','center')
uicontrol('Position',[280 80 80 30],'String','Done',...
    'Callback',@done);
uicontrol('Position',[280 30 80 30],'String','Cancel',...
    'Callback',@cancel);

if ~isempty(prop.numfiles)
    uicontrol('Position',[380 385 350 16],...
        'Style','text',...
        'String',numstr,...
        'ForegroundColor','r',...
        'HorizontalAlignment','center')
end

set(fig,'HandleVisibility','off')

uiwait(fig)

% Compute desired output.
switch prop.output
    case 'cell'
        out = full_file_picks;
    case 'struct'
        out = dir_picks(:);
    case 'char'
        out = char(full_file_picks);
    case 'cancel'
        out = 0;
end

% -------------------- Callback functions --------------------

    function add(varargin)
        values = get(navlist,'Value');
        for i = 1:length(values)
            dir_pick = fdir(values(i));
            pick = dir_pick.name;
            pick_full = fullfile(current_dir,pick);
            dir_pick.name = pick_full;
            if ~nodupes || ~any(strcmp(full_file_picks,pick_full))
                file_picks{end + 1} = pick;
                full_file_picks{end + 1} = pick_full;
                dir_picks(end + 1) = dir_pick;
            end
        end
        if show_full_path
            set(pickslist,'String',full_file_picks,'Value',[]);
        else
            set(pickslist,'String',file_picks,'Value',[]);
        end
        set([removebut,moveupbut,movedownbut],'Enable','off');
    end

    function remove(varargin)
        values = get(pickslist,'Value');
        file_picks(values) = [];
        full_file_picks(values) = [];
```

```
        dir_picks(values) = [];
        top = get(pickslist,'ListboxTop');
        num_above_top = sum(values < top);
        top = top - num_above_top;
        num_picks = length(file_picks);
        new_value = min(min(values) - num_above_top,num_picks);
        if num_picks == 0
            new_value = [];
            set([removebut,moveupbut,movedownbut],'Enable','off')
        end
        if show_full_path
            set(pickslist,'String',full_file_picks,'Value',new_value,...
                'ListboxTop',top)
        else
            set(pickslist,'String',file_picks,'Value',new_value,...
                'ListboxTop',top)
        end
    end

    function open(varargin)
        values = get(navlist,'Value');
        if fdir(values).isdir
            if strcmp(fdir(values).name,'.')
                return
            elseif strcmp(fdir(values).name,'..')
                set(dir_popup,'Value',min(2,length(path_cell)))
                dirpopup();
                return
            end
            current_dir = fullfile(current_dir,fdir(values).name);
            history{end+1} = current_dir;
            history = unique(history);
            hist_menus = make_history_cm(hist_cb,hist_cm,hist_menus,...
                history);
            full_filter = fullfile(current_dir,filter);
            path_cell = path2cell(current_dir);
            fdir = filtered_dir(full_filter,re_filter);
            filenames = {fdir.name}';
            filenames = annotate_file_names(filenames,fdir);
            set(dir_popup,'String',path_cell(end:-1:1),'Value',1)
            set(pathbox,'String',current_dir)
            set(navlist,'ListboxTop',1,'Value',[],'String',filenames)
            set(addbut,'Enable','off')
            set(openbut,'Enable','off')
        end
    end

    function clicknav(varargin)
        value = get(navlist,'Value');
        nval = length(value);
        dbl_click_fcn = @add;
        switch nval
            case 0
                set([addbut,openbut],'Enable','off')
            case 1
                set(addbut,'Enable','on');
                if fdir(value).isdir
                    set(openbut,'Enable','on')
                    dbl_click_fcn = @open;
                else
                    set(openbut,'Enable','off')
                end
            otherwise
                set(addbut,'Enable','on')
                set(openbut,'Enable','off')
        end
        if strcmp(get(fig,'SelectionType'),'open')
            dbl_click_fcn();
        end
    end

    function clickpicks(varargin)
        value = get(pickslist,'Value');
        if isempty(value)
            set([removebut,moveupbut,movedownbut],'Enable','off')
        else
            set(removebut,'Enable','on')
            if min(value) == 1
                set(moveupbut,'Enable','off')
            else
                set(moveupbut,'Enable','on')
            end
            if max(value) == length(file_picks)
                set(movedownbut,'Enable','off')
            else
                set(movedownbut,'Enable','on')
```

```
            end
        end
        if strcmp(get(fig,'SelectionType'),'open')
            remove();
        end
    end

    function dirpopup(varargin)
        value = get(dir_popup,'Value');
        len = length(path_cell);
        path_cell = path_cell(1:end-value+1);
        if ispc && value == len
            current_dir = '';
            full_filter = filter;
            fdir = struct('name',getdrives,'date',datestr(now),...
                'bytes',0,'isdir',1);
        else
            current_dir = cell2path(path_cell);
            history{end+1} = current_dir;
            history = unique(history);
            hist_menus = make_history_cm(hist_cb,hist_cm,hist_menus,...
                history);
            full_filter = fullfile(current_dir,filter);
            fdir = filtered_dir(full_filter,re_filter);
        end
        filenames = {fdir.name}';
        filenames = annotate_file_names(filenames,fdir);
        set(dir_popup,'String',path_cell(end:-1:1),'Value',1)
        set(pathbox,'String',current_dir)
        set(navlist,'String',filenames,'Value',[])
        set(addbut,'Enable','off')
    end

    function change_path(varargin)
        proposed_path = get(pathbox,'String');
        % Process any directories named '..'.
        proposed_path_cell = path2cell(proposed_path);
        ddots = strcmp(proposed_path_cell,'..');
        ddots(find(ddots) - 1) = true;
        proposed_path_cell(ddots) = [];
        proposed_path = cell2path(proposed_path_cell);
        % Check for existance of directory.
        if ~exist(proposed_path,'dir')
            uiwait(errordlg(['Directory "',proposed_path,...
                '" does not exist.'],'','modal'))
            return
        end
        current_dir = proposed_path;
        history{end+1} = current_dir;
        history = unique(history);
        hist_menus = make_history_cm(hist_cb,hist_cm,hist_menus,history);
        full_filter = fullfile(current_dir,filter);
        path_cell = path2cell(current_dir);
        fdir = filtered_dir(full_filter,re_filter);
        filenames = {fdir.name}';
        filenames = annotate_file_names(filenames,fdir);
        set(dir_popup,'String',path_cell(end:-1:1),'Value',1)
        set(pathbox,'String',current_dir)
        set(navlist,'String',filenames,'Value',[])
        set(addbut,'Enable','off')
        set(openbut,'Enable','off')
    end

    function showfullpath(varargin)
        show_full_path = get(viewfullpath,'Value');
        if show_full_path
            set(pickslist,'String',full_file_picks)
        else
            set(pickslist,'String',file_picks)
        end
    end

    function removedupes(varargin)
        nodupes = get(remove_dupes,'Value');
        if nodupes
            num_picks = length(full_file_picks);
            [unused,rev_order] = unique(full_file_picks(end:-1:1));
            order = sort(num_picks + 1 - rev_order);
            full_file_picks = full_file_picks(order);
            file_picks = file_picks(order);
            if show_full_path
                set(pickslist,'String',full_file_picks,'Value',[])
            else
                set(pickslist,'String',file_picks,'Value',[])
            end
            set([removebut,moveupbut,movedownbut],'Enable','off')
```

```
        end
    end

    function moveup(varargin)
        value = get(pickslist,'Value');
        set(removebut,'Enable','on')
        n = length(file_picks);
        omega = 1:n;
        index = zeros(1,n);
        index(value - 1) = omega(value);
        index(setdiff(omega,value - 1)) = omega(setdiff(omega,value));
        file_picks = file_picks(index);
        full_file_picks = full_file_picks(index);
        value = value - 1;
        if show_full_path
            set(pickslist,'String',full_file_picks,'Value',value)
        else
            set(pickslist,'String',file_picks,'Value',value)
        end
        if min(value) == 1
            set(moveupbut,'Enable','off')
        end
        set(movedownbut,'Enable','on')
    end

    function movedown(varargin)
        value = get(pickslist,'Value');
        set(removebut,'Enable','on')
        n = length(file_picks);
        omega = 1:n;
        index = zeros(1,n);
        index(value + 1) = omega(value);
        index(setdiff(omega,value + 1)) = omega(setdiff(omega,value));
        file_picks = file_picks(index);
        full_file_picks = full_file_picks(index);
        value = value + 1;
        if show_full_path
            set(pickslist,'String',full_file_picks,'Value',value)
        else
            set(pickslist,'String',file_picks,'Value',value)
        end
        if max(value) == n
            set(movedownbut,'Enable','off')
        end
        set(moveupbut,'Enable','on')
    end

    function togglefilter(varargin)
        value = get(showallfiles,'Value');
        if value
            filter = '*';
            re_filter = '';
            set([filter_ed,refilter_ed],'Enable','off')
        else
            filter = get(filter_ed,'String');
            re_filter = get(refilter_ed,'String');
            set([filter_ed,refilter_ed],'Enable','on')
        end
        full_filter = fullfile(current_dir,filter);
        fdir = filtered_dir(full_filter,re_filter);
        filenames = {fdir.name}';
        filenames = annotate_file_names(filenames,fdir);
        set(navlist,'String',filenames,'Value',[])
        set(addbut,'Enable','off')
    end

    function setfilspec(varargin)
        filter = get(filter_ed,'String');
        if isempty(filter)
            filter = '*';
            set(filter_ed,'String',filter)
        end
        % Process file spec if a subdirectory was included.
        [p,f,e] = fileparts(filter);
        if ~isempty(p)
            newpath = fullfile(current_dir,p,'');
            set(pathbox,'String',newpath)
            filter = [f,e];
            if isempty(filter)
                filter = '*';
            end
            set(filter_ed,'String',filter)
            change_path();
        end
        full_filter = fullfile(current_dir,filter);
        fdir = filtered_dir(full_filter,re_filter);
```

```
        filenames = {fdir.name}';
        filenames = annotate_file_names(filenames,fdir);
        set(navlist,'String',filenames,'Value',[])
        set(addbut,'Enable','off')
    end

    function setrefilter(varargin)
        re_filter = get(refilter_ed,'String');
        fdir = filtered_dir(full_filter,re_filter);
        filenames = {fdir.name}';
        filenames = annotate_file_names(filenames,fdir);
        set(navlist,'String',filenames,'Value',[])
        set(addbut,'Enable','off')
    end

    function done(varargin)
        % Optional shortcut: click on a file and press 'Done'.
%         if isempty(full_file_picks) && strcmp(get(addbut,'Enable'),'on')
%             add();
%         end
        numfiles = length(full_file_picks);
        if ~isempty(prop.numfiles)
            if numfiles < prop.numfiles(1)
                msg = {'Too few files selected.',numstr};
                uiwait(errordlg(msg,'','modal'))
                return
            elseif numfiles > prop.numfiles(end)
                msg = {'Too many files selected.',numstr};
                uiwait(errordlg(msg,'','modal'))
                return
            end
        end
        delete(fig)
    end

    function cancel(varargin)
        prop.output = 'cancel';
        delete(fig)
    end

    function history_cb(varargin)
        current_dir = history{varargin{3}};
        full_filter = fullfile(current_dir,filter);
        path_cell = path2cell(current_dir);
        fdir = filtered_dir(full_filter,re_filter);
        filenames = {fdir.name}';
        filenames = annotate_file_names(filenames,fdir);
        set(dir_popup,'String',path_cell(end:-1:1),'Value',1)
        set(pathbox,'String',current_dir)
        set(navlist,'ListboxTop',1,'Value',[],'String',filenames)
        set(addbut,'Enable','off')
        set(openbut,'Enable','off')
    end
end


% -------------------- Subfunctions --------------------

function c = path2cell(p)
% Turns a path string into a cell array of path elements.
c = strread(p,'%s','delimiter','\\/');
if ispc
    c = [{'My Computer'};c];
else
    c = [{filesep};c(2:end)];
end
end


function p = cell2path(c)
% Turns a cell array of path elements into a path string.
if ispc
    p = fullfile(c{2:end},'');
else
    p = fullfile(c{:},'');
end
end


function d = filtered_dir(full_filter,re_filter)
% Like dir, but applies filters and sorting.
p = fileparts(full_filter);
if isempty(p) && full_filter(1) == '/'
    p = '/';
end
if exist(full_filter,'dir')
```

```
    c = cell(0,1);
    dfiles = struct('name',c,'date',c,'bytes',c,'isdir',c);
else
    dfiles = dir(full_filter);
end
if ~isempty(dfiles)
    dfiles([dfiles.isdir]) = [];
end
ddir = dir(p);
ddir = ddir([ddir.isdir]);
% Additional regular expression filter.
if nargin > 1 && ~isempty(re_filter)
    if ispc
        no_match = cellfun('isempty',regexpi({dfiles.name},re_filter));
    else
        no_match = cellfun('isempty',regexp({dfiles.name},re_filter));
    end
    dfiles(no_match) = [];
end
% Set navigator style:
%    1 => mix file and directory names
%    2 => means list all files before all directories
%    3 => means list all directories before all files
%    4 => same as 2 except put . and .. directories first
if isunix
    style = 4;
else
    style = 4;
end
switch style
    case 1
        d = [dfiles;ddir];
        [unused,index] = sort({d.name});
        d = d(index);
    case 2
        [unused,index1] = sort({dfiles.name});
        [unused,index2] = sort({ddir.name});
        d = [dfiles(index1);ddir(index2)];
    case 3
        [unused,index1] = sort({dfiles.name});
        [unused,index2] = sort({ddir.name});
        d = [ddir(index2);dfiles(index1)];
    case 4
        [unused,index1] = sort({dfiles.name});
        dot1 = find(strcmp({ddir.name},'.'));
        dot2 = find(strcmp({ddir.name},'..'));
        ddot1 = ddir(dot1);
        ddot2 = ddir(dot2);
        ddir([dot1,dot2]) = [];
        [unused,index2] = sort({ddir.name});
        d = [ddot1;ddot2;dfiles(index1);ddir(index2)];
end
end


function drives = getdrives
% Returns a cell array of drive names on Windows.
letters = char('A':'Z');
num_letters = length(letters);
drives = cell(1,num_letters);
for i = 1:num_letters
    if exist([letters(i),':\'],'dir');
        drives{i} = [letters(i),':'];
    end
end
drives(cellfun('isempty',drives)) = [];
end


function filenames = annotate_file_names(filenames,dir_listing)
% Adds a trailing filesep character to directory names.
fs = filesep;
for i = 1:length(filenames)
    if dir_listing(i).isdir
        filenames{i} = [filenames{i},fs];
    end
end
end


function hist_menus = make_history_cm(cb,hist_cm,hist_menus,history)
% Make context menu for history.
if ~isempty(hist_menus)
    delete(hist_menus)
end
num_hist = length(history);
```

```
hist_menus = zeros(1,num_hist);
for i = 1:num_hist
    hist_menus(i) = uimenu(hist_cm,'Label',history{i},...
        'Callback',{cb,i});
end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function welcome2LTPDA(varargin)
% 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

global timePeriod

timePeriod = 0;
screenSize = get(0,'ScreenSize');

try
   guiName = varargin{1}; guiSize = varargin{2};
catch
   guiName = ''; guiSize = [1 1 450 500];
end

introSize = [(screenSize(3)-guiSize(3))/2 , (screenSize(4)-guiSize(4))/2 , guiSize(3) , guiSize(4)];
introFig =
figure('Position',introSize,'Resize','off','Name',guiName,'NumberTitle','off','Toolbar','none','Menubar','none','Tag',
);


logoSize = [303,303];
introPosition = [(introSize(3)-logoSize(1))/2 , (introSize(4)-logoSize(2))/2 , logoSize];
logo = axes('Parent',introFig,'Units','pixels','Position',introPosition);
image(imread('ltp.jpg'),'Parent',logo,'AlphaData',imread('ltpalpha.gif'));
axis(logo,'off'); axis(logo,'square');
set(logo,'HandleVisibility','callback');

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');
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');

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');

    gif_image = 'loading.gif';
    delay_length = 0.1;
    gifSize = [23 23];
    axes('Parent',introFig,'Units','pixels','Position',[(introSize(3)-gifSize(1))/2 50 gifSize]);
    [handles.im,map] = imread(gif_image,'frames','all');
```

```
    handles.len = size(handles.im,4);
        handles.h1 = image(handles.im(:,:,:,1));
        colormap(map);
        axis normal;
        axis off;
    handles.guifig = gcf;
    handles.count = 1;
    handles.tmr = timer('TimerFcn',
{@TmrFcn,handles.guifig},'BusyMode','Queue','ExecutionMode','FixedRate','Period',delay_length);
    guidata(handles.guifig, handles);
    start(handles.tmr); %starts Timer
guidata(handles.guifig, handles);

set(loading,'DeleteFcn',{@DeleteFunction,handles.guifig});


function TmrFcn(src,event,handles)
handles = guidata(handles);
set(handles.h1,'CData',handles.im(:,:,:,handles.count)); %update the frame in the axis
handles.count = handles.count + 1; %increment to next frame

if handles.count > handles.len %if the last frame is achieved intialise to first frame
    handles.count = 1;
end
guidata(handles.guifig, handles);

loadingStr = get(findobj('Tag','loading'),'String');
switch loadingStr
   case 'loading.....', loadingStr = 'loading';
   otherwise, loadingStr = [loadingStr,'.'];
end

if timePeriod == 2, set(findobj('Tag','loading'),'String',loadingStr); timePeriod = 0;
else timePeriod = timePeriod +1;
end
end


function DeleteFunction(src,event,handles)
handles = guidata(handles);
clear global timePeriod
try stop(handles.tmr);delete(handles.tmr); catch end
end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 graphic uitable 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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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.ListSelecti


 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:
       <a
href="http://java.sun.com/docs/books/tutorial/uiswing/components/table.html">http://java.sun.com/
```

```
 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

 Notes:
    1. Some (very few) JTable features are inconsistent or unavailable in different jave
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 ^

```
function [mtable, buttons] = createTable(pnContainer,headers,data,buttonsFlag,varargin)
% 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.ListSelecti
```

```
%
% 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:
%       <a href="http://java.sun.com/docs/books/tutorial/uiswing/components/table.html">http://java.sun.com/

%
% 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
%
% Notes:
%    1. Some (very few) JTable features are inconsistent or unavailable in different jave 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 $
%
%

% License to use and modify this code is granted freely to all interested, as long as the original author is
% referenced and attributed as such. The original author maintains the right to be solely associated with this work.

% Programmed and Copyright by Yair M. Altman: altmany(at)gmail.com
% $Revision: 1.2 $  $Date: 2008/03/01 11:46:46 $

  %try
      % Ensure that java swing is enabled...
      if ~usejava('swing')
          error('createTable:NeedSwing','Java tables require Java Swing.');
      end

      % Create a panel spanning entire figure area, if panel handle was not supplied
      if (nargin < 1) || isempty(pnContainer) || ~ishandle(pnContainer)
          pnContainer = uipanel('parent',gcf,'tag','TablePanel');
      end
      pnContainerPos = getpixelposition(pnContainer,1);
      if isa(handle(pnContainer), 'figure')
          pnContainerPos(1:2) = 0;
      end

      % Get handle to parent figure
      hFig = ancestor(pnContainer,'figure');

      % Determine whether table manipulation buttons are requested
      if nargin < 4 || isempty(buttonsFlag) || ~(isnumeric(buttonsFlag) || islogical(buttonsFlag))
          if nargin >= 4,  varargin = {buttonsFlag, varargin{:}};  end
          buttonsFlag = true;
      end
      if buttonsFlag
          margins = [1,30,0,-30];  % With buttons
      else
          margins = [1,1,0,0];   % No buttons
      end

      % Get the uitable's required position within pnContainer
      tablePosition = pnContainerPos + margins;    % Relative to the figure

      % Set default header names, if not supplied
      if nargin < 2
          headers = {'A','B','C'};  % 3 columns by default
      elseif isempty(headers)
          headers = cell(1,size(data,2));
      elseif ischar(headers)
          headers = {headers};
      end

      % 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.mwswing.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> <b>Click</b> to sort up;
<b>Shift-click</b> to sort down<br> <b>Ctrl-click</b> (or <b>Ctrl-Shift-click</b>) to
sort secondary <br> <b>Click again</b> to change sort direction<br> <b>Click a
third time</b> to return to unsorted view<br> <b>Right-click</b> to select entire
column</html>');
0205        else
0206            % Set the header tooltip (no sorting instructions...)
0207            jtable.getTableHeader.setToolTipText('<html> <b>Click</b> to select
entire column<br> <b>Ctrl-click</b> (or <b>Shift-click</b>) to select multiple
columns </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
```

```
                % TODO: We should really use calls to awtinvoke() instead, though...
                pause(0.05);
            end
            if (length(varargin) > argIdx)   % ensure the arg value is there...
                varargin{argIdx}(1) = upper(varargin{argIdx}(1));  % property names always start with capital letters...
                propMethodName = ['set' varargin{argIdx}];
                if ismethod(mtable,propMethodName)
                    set(mtable,varargin{argIdx},varargin{argIdx+1});
                else
                    %javaMethod(propMethodName, jtable, varargin{argIdx+1});
                    set(jtable,varargin{argIdx},varargin{argIdx+1});
                end
            end
        end  % for argIdx

        % Create table manipulation buttons
        if buttonsFlag
            buttons = createManipulationButtons(pnContainer,mtable);
        else
            buttons = [];
        end
    %catch
        % Insert your code here
        %handleError;
    %end


%% --- Executes on button press in btInsert.
function buttons = createManipulationButtons(pnContainer, mtable)
% pnContainer  handle to container uipanel
% mtable       handle to mtable (Matlab) object
    %try
        btAppendRow = uicontrol('tag','btTableAppendRow', 'callback',@btTableAppendRow_Callback, 'position',  [10,5,60,20], 'string','Append', 'parent',pnContainer, 'userdata',mtable);
        btInsertRow = uicontrol('tag','btTableInsertRow', 'callback',@btTableInsertRow_Callback, 'position',  [80,5,60,20], 'string','Insert', 'parent',pnContainer, 'userdata',mtable);
        btDeleteRow = uicontrol('tag','btTableDeleteRow', 'callback',@btTableDeleteRow_Callback, 'position', [150,5,60,20], 'string','Delete', 'parent',pnContainer, 'userdata',mtable);
        btDeleteAll = uicontrol('tag','btTableDeleteAll', 'callback',@btTableDeleteAll_Callback, 'position', [220,5,60,20], 'string','Delete All', 'parent',pnContainer, 'userdata',mtable);
        btPrintAll  = uicontrol('tag','btTablePrintAll', 'callback',@btTablePrintAll_Callback,  'position', [290,5,60,20], 'string','Print', 'parent',pnContainer, 'userdata',mtable);
        buttons = [btInsertRow btAppendRow btDeleteRow btDeleteAll btPrintAll];
        if mtable.getNumRows < 1
            setVisibility(pnContainer, 'off');
        end
    %catch
        % Insert your code here
        %handleError;
    %end


%% --- Executes on button press in btInsert.
% Insert a new row immediately before the current row
function btTableInsertRow_Callback(hObject, eventdata, handles)  %#ok
% hObject    handle to btTableInsertRow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    %try
        mtable = get(hObject,'userdata');
        jtable = mtable.getTable;

        % Stop any current editing
        stopEditing(jtable);

        % Insert the new row immediately before the current row
        newRowData = cell(1,mtable.getNumColumns);  % empty data
        mtable.getTableModel.insertRow(max(0,jtable.getSelectedRow), newRowData);
    %catch
        % Insert your code here
        %handleError;
    %end


%% --- Executes on button press in btAppend.
% Insert a new row as the last row in the table
function btTableAppendRow_Callback(hObject, eventdata, handles)  %#ok
% hObject    handle to btTableAppendRow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% 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
```

```
% handles    structure with handles and user data (see GUIDATA)
  persistent hExcel
  try
      mtable = get(hObject,'userdata');

      % Try to open an Excel COM server
      % Note: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odc_2003_ta/html/odc_landoffice03_vba.asp
      try
          % try to reuse an existing (pre-opened) COM server
          % If we can't access the ActiveX parent, it means it's closed
          parent = hExcel.parent;  %#ok
      catch
          hExcel = actxserver('excel.application');
      end

      % Try to open the requested document
      hExcel.Workbooks.Add;

      % Format field headers
      headers = cell(mtable.getColumnNames)';
      if ~isempty(headers)
          hExcel.Range(['A1:' n2a(length(headers)) '1']).Select;
          hExcel.Selection.Value = headers;
          hExcel.Selection.Font.Bold = true;                    % bold
          hExcel.Selection.Font.Color = hex2dec('0000FF');      % Red
          hExcel.Selection.Border.Item(4).Weight = 3;           % underline
          hExcel.Selection.Cells.HorizontalAlignment = -4108;   % =xlCenter
      end

      % Set the data from the table
      data = cell(mtable.data);
      if ~isempty(headers)
          hExcel.Range(['A2:' n2a(size(data,2)) num2str(1+size(data,1))]).Select;
          hExcel.Selection.Value = data;
      end

      % TODO: Change checkbox fields to boolean (TRUE/empty)

      % Other formats
      %hExcel.Cells.HorizontalAlignment = -4108;  % =xlCenter  % TODO: preserve original jtable column alignment
      hExcel.Cells.EntireColumn.AutoFit;
      hExcel.ActiveSheet.DisplayRightToLeft = false;
      set(hExcel.ActiveSheet.PageSetup, 'LeftMargin',  hExcel.InchesToPoints(0.1), ...
                                        'RightMargin', hExcel.InchesToPoints(0.1), ...
                                        'HeaderMargin',hExcel.InchesToPoints(0), ...
                                        'FooterMargin',hExcel.InchesToPoints(0.1), ...
                                        'TopMargin',120, ...
                                        'FitToPagesWide',1, ...
                                        'FitToPagesTall',1, ...
                                        'Orientation','xlPortrait', ...
                                        'LeftHeader','&D  &T', ...
                                        'CenterHeader',char(mtable.getTable.getName), ...
                                        'RightHeader','&G');

      % Send to printer
      hExcel.ActiveWindow.SelectedSheets.PrintOut;

      % Close the workbook
      invoke(hExcel.ActiveWindow,'close',false);
  catch
      % Insert your code here
      %handleError;
      err = lasterror;
      try  invoke(hExcel.ActiveWindow,'close',false);  catch  end;  % just in case of a printing error
      rethrow(err);
  end


%% --- Convert col # format to 'A','B','C','AA',... format
% Thanks Brett Shoelson, via CSSM
function colStr = n2a(c)
  t = [floor((c-1)/26)+64, rem(c-1,26)+65];
  if (t(1)<65), t(1) = []; end
  colStr = char(t);


%% --- Executes on button press in btInsert.
function stopEditing(jtable)
  %try
      component = jtable.getEditorComponent;
      if ~isempty(component)
          event = javax.swing.event.ChangeEvent(component);
```

```
            jtable.editingStopped(event);
        end
    %catch
        % Insert your code here
        %handleError;
    %end


%% --- Utility function to set visibility of row manipulation buttons
function setVisibility(hObject, enableStr)
% hObject    handle to some element within the figure
% enableStr  'on' or 'off'
    %try
        hParent = ancestor(hObject,'figure');
        set(findall(hParent,'tag','btTableInsertRow'),'enable',enableStr);
        set(findall(hParent,'tag','btTableDeleteRow'),'enable',enableStr);
        set(findall(hParent,'tag','btTableDeleteAll'),'enable',enableStr);
        set(findall(hParent,'tag','btTablePrintAll'), 'enable',enableStr);
    %catch
        % Insert your code here
        %handleError;
    %end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

muitable

PURPOSE ^

UITABLE creates a two dimensional graphic uitable component in a figure window.

SYNOPSIS ^

function [table, container] = muitable(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 ^

```
function [table, container] = muitable(varargin)

% 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 $
%


%   Copyright 2002-2006 The MathWorks, Inc.
%   $Revision: 1.3 $  $Date: 2008/02/26 08:54:43 $

%   Release: R14. This feature will not work in previous versions of MATLAB.

% Setup and P-V parsing

error(javachk('awt'));
error(nargoutchk(0,2,nargout));

parent = [];
numargs = nargin;

datastatus=false; columnstatus=false;
rownum = 1; colnum = 1; % Default to a 1x1 table.
position = [20 20 200 200];
combo_box_found = false;
check_box_found = false;

import com.mathworks.hg.peer.UitablePeer;

if (numargs > 0 && isscalar(varargin{1}) && ishandle(varargin{1}) && ...
        isa(handle(varargin{1}), 'figure'))
    parent = varargin{1};
    varargin = varargin(2:end);
    numargs = numargs - 1;
end

if (numargs > 0 && isscalar(varargin{1}) &&  ishandle(varargin{1}))
    if ~isa(varargin{1}, 'javax.swing.table.DefaultTableModel')
        error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter: ', varargin{1}]);
    end
    data_model = varargin{1};
    varargin = varargin(2:end);
    numargs = numargs - 1;

elseif ((numargs > 1) && isnumeric(varargin{1}) && isnumeric(varargin{2}))
    if(isnumeric(varargin{1}) && isnumeric(varargin{2}))
        rownum = varargin{1};
        colnum = varargin{2};

        varargin = varargin(3:end);
        numargs = numargs-2;
    else
        error('MATLAB:uitable:InputMustBeScalar', 'When using UITABLE numrows and numcols have to be numeric scalars.')
    end

elseif ((numargs > 1) && isequal(size(varargin{2},1), 1) && iscell(varargin{2}))
    if (size(varargin{1},2) == size(varargin{2},2))
        if (isnumeric(varargin{1}))
            varargin{1} = num2cell(varargin{1});
        end
    else
        error('MATLAB:uitable:MustMatchInfo', 'Number of column names must match number of columns in data');
    end
    data = varargin{1};     datastatus        = true;
    coln = varargin{1+1};   columnstatus      = true;

    varargin = varargin(3:end);
    numargs = numargs-2;
end

for i = 1:2:numargs-1
    if (~ischar(varargin{i}))
        error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter: ', varargin{i}]);
    end
    switch lower(varargin{i})
        case 'data'
            if (isnumeric(varargin{i+1}))
                varargin{i+1} = num2cell(varargin{i+1});
            end
```

```
            data        = varargin{i+1};
            datastatus  = true;

        case 'columnnames'
            if(iscell(varargin{i+1}))
                coln            = varargin{i+1};
                columnstatus    = true;
            else
                error('MATLAB:uitable:InvalidCellArray', 'When using UITABLE Column data should be 1xn cell array')
            end

        case 'numrows'
            if (isnumeric(varargin{i+1}))
                rownum = varargin{i+1};
            else
                error('MATLAB:uitable:NumrowsMustBeScalar', 'numrows has to be a scalar')
            end

        case 'numcolumns'
            if (isnumeric(varargin{i+1}))
                colnum = varargin{i+1};
            else
                error('MATLAB:uitable:NumcolumnsMustBeScalar', 'numcolumns has to be a scalar')
            end

        case 'gridcolor'
            if (ischar(varargin{i+1}))
                gridcolor = varargin{i+1};
            else if (isnumeric(varargin{i+1}) && (numel(varargin{i+1}) == 3))
                    gridcolor = varargin{i+1};
                else
                    error('MATLAB:uitable:InvalidString', 'gridcolor has to be a valid string')
                end
            end

        case 'rowheight'
            if (isnumeric(varargin{i+1}))
                rowheight = varargin{i+1};
            else
                error('MATLAB:uitable:RowheightMustBeScalar', 'rowheight has to be a scalar')
            end

        case 'parent'
            if ishandle(varargin{i+1})
                parent = varargin{i+1};
            else
                error('MATLAB:uitable:InvalidParent', 'parent must be a valid handle')
            end

        case 'position'
            if (isnumeric(varargin{i+1}))
                position = varargin{i+1};
            else
                error('MATLAB:uitable:InvalidPosition', 'position has to be a 1x4 numeric array')
            end

        case 'columnwidth'
            if (isnumeric(varargin{i+1}))
                columnwidth = varargin{i+1};
            else
                error('MATLAB:uitable:ColumnwidthMustBeScalar', 'columnwidth has to be a scalar')
            end
        otherwise
            error('MATLAB:uitable:UnrecognizedParameter', ['Unrecognized parameter: ', varargin{i}]);
    end
end

% ---combo/check box detection--- %
if (datastatus)
    if (iscell(data))
        rownum = size(data,1);
        colnum = size(data,2);
        combo_count =0;
        check_count = 0;
        combo_box_data   = num2cell(zeros(1, colnum));
        combo_box_column = zeros(1, colnum);
        check_box_column = zeros(1, colnum);
        for j = 1:rownum
```

```
            for k = 1:colnum
                if (iscell(data{j,k}))
                    combo_box_found = true;
                    combo_count = combo_count + 1;
                    combo_box_data{combo_count} = data{j,k};
                    combo_box_column(combo_count ) = k;
                    dc = data{j,k};
                    data{j,k} = dc{1};
                else
                    if(islogical(data{j,k}))
                        check_box_found = true;
                        check_count = check_count + 1;
                        check_box_column(check_count) = k;
                    end
                end
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check the validity of the parent and/or create a figure.
if isempty(parent)
    parent = gcf; % Get the current figure. Create one if not available
end

if ( columnstatus && datastatus )
    if(size(data,2) ~= size(coln,2))
        error('MATLAB:NeedSameNumberColumns', 'Number of columns in both Data and ColumnNames should match');
    end
elseif ( ~columnstatus && datastatus )
    for i=1:size(data,2)
        coln{i} = num2str(i);
    end
    columnstatus = true;
elseif ( columnstatus && ~datastatus)
    error('MATLAB:uitable:NoDataProvided', 'No Data provided along with ColumnNames');
end

if (~exist('data_model', 'var'))
    data_model = javax.swing.table.DefaultTableModel;
end
if exist('rownum', 'var')
    data_model.setRowCount(rownum);
end
if exist('colnum', 'var')
    data_model.setColumnCount(colnum);
end

table_h= UitablePeer(data_model);

% We should have valid data and column names here.
if (datastatus), table_h.setData(data); end;
if (columnstatus), table_h.setColumnNames(coln); end;

if (combo_box_found),
    for i=1:combo_count
        table_h.setComboBoxEditor(combo_box_data(i), combo_box_column(i));
    end
end
if (check_box_found),
    for i = 1: check_count
        table_h.setCheckBoxEditor(check_box_column(i));
    end
end

% pass the specified parent and let javacomponent decide its validity.
[obj, container] = javacomponent(table_h, position, parent);
% javacomponent returns a UDD handle for the java component passed in.
table = obj;

% Have to do a drawnow here to make the properties stick. Try to restrict
% the drawnow call to only when it is absolutely required.
flushed = false;
if exist('gridcolor', 'var')
    pause(.1); drawnow;
    flushed = true;
    table_h.setGridColor(gridcolor);
end
if exist('rowheight', 'var')
    if (~flushed)
        drawnow;
    end
    table_h.setRowHeight(rowheight);
end
```

```
if exist('columnwidth', 'var')
    table_h.setColumnWidth(columnwidth);
end;



% Add a predestroy listener so we can call cleanup on the table.
% addlistener(table, 'ObjectBeingDestroyed', {@componentDelete});

function componentDelete(src, evd)                                        %#ok
% Clean up the table here so it disengages all its internal listeners.
src.cleanup;
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = repogui(varargin)

% 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 $
%

% addpath('callbacks');

%% Check if I exist already
id = findobj('Tag', 'LTPDARepomainfig');
if ~isempty(id)
  figure(id)
  return
end

%% Some initial setup

Gproperties.Gcol    = [240 240 240]/255;
Gproperties.Gwidth  = 800;
Gproperties.Gheight = 650;
Gproperties.Gborder = 10;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');

```

```
Gproperties.Screen   = get(0,'screensize');
Gproperties.Gposition = [Gproperties.Screen(3)/2-Gproperties.Gwidth/2 ...
  Gproperties.Screen(4)/2-Gproperties.Gheight/2 ...
  Gproperties.Gwidth...
  Gproperties.Gheight];

%  Initialize and hide the GUI as it is being constructed.
mainfig = figure('Name', 'LTPDA Repository GUI',...
  'NumberTitle', 'off',...
  'Visible','off',...
  'Position',Gproperties.Gposition,...
  'Color', Gproperties.Gcol,...
  'Toolbar', 'none',...
  'MenuBar', 'none',...
  'Resize', 'off',...
  'Tag', 'LTPDARepomainfig');

% Set mainfig callbacks
set(mainfig, 'CloseRequestFcn', {'cb_mainfigClose', mainfig});

% Set Application data
setappdata(mainfig, 'Gproperties', Gproperties);
setappdata(mainfig, 'connection', []);
setappdata(mainfig, 'Nconditions', 0);
setappdata(mainfig, 'tableSelected', 1);

%% Add components

Tleft   = Gproperties.Gborder/Gproperties.Gwidth;
Theight = 1-2*Gproperties.Gborder/Gproperties.Gheight - 0.07;
Tbottom = (1 - Gproperties.Gborder/Gproperties.Gheight - Theight);
Twidth  = 1-2*Gproperties.Gborder/Gproperties.Gwidth;

htab = uitabpanel(...
  'Parent',mainfig,...
  'Style','lefttop',...
  'Units','normalized',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Fontsize', fontsize,...
  'FrameBackgroundColor',Gproperties.Gcol,...
  'FrameBorderType','etchedout',...
  'Title',{'Connection','Submit','Query', 'Retrieve'},...%, 'Quick Look'},...
  'HorizontalAlignment','left',...
  'FontWeight','bold',...
  'TitleBackgroundColor',Gproperties.Gcol,...
  'TitleForegroundColor','k',...
  'TitleHighlightColor', [1 0 0],...
  'PanelBackgroundColor',Gproperties.Gcol,...
  'PanelBorderType','line');

hpanel = getappdata(htab,'panels');
setappdata(mainfig, 'panels', hpanel);

%-------- Connection Panel -------------
cp = create_connection_panel(hpanel(1), Gproperties);
%-------- Submit Panel -------------
sp = create_submit_panel(hpanel(2), Gproperties);
%-------- Query Panel -------------
qp = create_query_panel(hpanel(3), Gproperties);
%-------- Query Panel -------------
rp = create_retrieve_panel(hpanel(4), Gproperties);
%-------- Quick Look Panel -------------
% rp = create_quicklook_panel(hpanel(5), Gproperties);

Ppos    = get(mainfig, 'Position');
%-- Status Txt
Sleft   = Gproperties.Gborder;
Sbottom = Gproperties.Gborder + 10;
Swidth  = 60;
Sheight = 20;

sth = uicontrol(mainfig,'Style','text',...
  'String','Status:',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Sbottom Swidth Sheight]);
%-- Status bar
pos = get(sth, 'Position');
Sleft   = 4*Gproperties.Gborder + pos(2);
Swidth  = Gproperties.Gwidth  - Gproperties.Gborder - Sleft;
Sheight = 30;
Sbottom = Gproperties.Gborder;
sth = uicontrol(mainfig,'Style','text',...
  'String','not connected',...
  'Fontsize', fontsize,...
```

```
  'BackgroundColor', 'w',...
  'ForegroundColor', 'r', ...
  'HorizontalAlignment', 'left',...
  'Max', 100,...
  'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'statusTxt');

%% Start the GUI

% Make the GUI visible.
set(mainfig,'Visible','on')

h = findobj(mainfig, 'Tag', 'statusTxt');
get(h, 'String')

end

%-------------------------------------------------------------------------
% Create the quick look panel
%
function panel = create_quicklook_panel(panel, Gproperties)

Ppos    = get(panel, 'Position');
mainfig = findobj('Tag', 'LTPDARepomainfig');

%-- Text string
Sleft   = 10;
Sheight = 20;
Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
Swidth  = 200;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
sth = uicontrol(panel,'Style','text',...
  'String','LTPDA Objects',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'center',...
  'Position',[Sleft Sbottom Swidth Sheight]);

%-- Workspace list of LTPDA objects
Lleft   = 10;
Lheight = 350;
Lbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Lheight;
Lwidth  = Swidth;

lbh = uicontrol(panel,'Style','listbox',...
  'String',{' '},...
  'Value',1,...
  'BackgroundColor', 'w',...
  'Fontsize', fontsize,...
  'Max', 1000,...
  'Position',[Lleft Lbottom Lwidth Lheight],...
  'Tag', 'workspaceObjsListQL');

% Set callback
set(lbh, 'Callback', {'cb_workspaceObjsListQL', lbh, mainfig});

% fill list
objs = getWorkspaceObjs();
setWorkspaceObjsListQL(objs);

%-- Refresh button
pos     = get(lbh, 'Position');
Bleft   = Lleft;
Bheight = 20;
Bwidth  = 100;
Bbottom = pos(2) - Bheight - 10;
pbh = uicontrol(panel,'Style','pushbutton','String','Refresh list',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Tag', 'refreshListBtnQL');
set(pbh, 'Callback', {'cb_refreshObjsListQL', pbh, mainfig});


%-- Info panel
pos = get(lbh, 'Position');
Sleft   = 10 + pos(1) + pos(3);
Sheight = 150;
Sbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Sheight;
Swidth  = 550;
iph = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'ForegroundColor', 'b',...
  'HorizontalAlignment', 'left',...
  'Enable', 'on',...
  'Max', 1000,...
```

```
  'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'ltpdaObjInfoQL');

%-- Text string
pos = get(iph, 'Position');
Sheight = 20;
Sbottom = pos(2) - Sheight - 20;
Swidth  = 300;
sth = uicontrol(panel,'Style','text',...
  'String','Class Methods',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Sbottom Swidth Sheight]);

%-- List of class methods
pos = get(lbh, 'Position');
Lleft   = pos(1) + pos(3) + 10;
Lheight = 150;
Lbottom = pos(2);
Lwidth  = 150;

lbh = uicontrol(panel,'Style','listbox',...
  'String',{' '},...
  'Value',1,...
  'BackgroundColor', 'w',...
  'Fontsize', fontsize,...
  'Max', 1000,...
  'Position',[Lleft Lbottom Lwidth Lheight],...
  'Tag', 'classMethodList');

% Set callback
set(lbh, 'Callback', {'cb_classMethodList', lbh, mainfig});

% fill list
fillClassMethodsList();

%------------ Help panel
pos = get(lbh, 'Position');
Sleft   = pos(1) + pos(3) + 10;
Sheight = 150;
Sbottom = pos(2);
Swidth  = 550;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'ForegroundColor', 'b',...
  'HorizontalAlignment', 'left',...
  'Enable', 'on',...
  'Max', 1000,...
  'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'methodInfoPanel');


end

%-------------------------------------------------------------------------------
% Create the retrieve panel
%
function panel = create_retrieve_panel(panel, Gproperties)

Ppos    = get(panel, 'Position');
mainfig = findobj('Tag', 'LTPDARepomainfig');
ALLHEIGHT = 25;

%-- Text string
Sleft   = 10;
Sheight = ALLHEIGHT;
Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
Swidth  = 300;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
Osth = uicontrol(panel,'Style','text',...
  'String','Retrieve object IDs',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Sbottom Swidth Sheight]);

%-- ID entry
pos     = get(Osth, 'Position');
Sleft   = 10;
Sheight = 250;
Swidth  = 250;
Sbottom = pos(2) - 20 - Sheight;

```

```
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 10,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'retrieveIDsTxt');

%-- Prefix entry
pos     = get(sth, 'Position');
Sleft   = 10;
Sheight = ALLHEIGHT;
Sbottom = pos(2) - 20 - Sheight;
Swidth  = 50;
sth = uicontrol(panel,'Style','text',...
  'String','Prefix',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Sbottom Swidth Sheight]);


Sleft   = Sleft + Swidth + 20;
Sheight = ALLHEIGHT;
Swidth  = 150;
sth = uicontrol(panel,'Style','edit',...
  'String','obj',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'objPrefixTxt');

%-- Append obj type
pos = get(sth, 'Position');
Sleft   = 10;
Swidth  = 150;
Sheight = ALLHEIGHT;
Sbottom = pos(2) - 20 - Sheight;
cbh = uicontrol(panel,'Style','checkbox',...
  'String','Append object type',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'Value',1,'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'appendObjTypeChk');

%-- Import Btn

Bleft   = 10;
Bheight = ALLHEIGHT;
Bwidth  = 100;
Bbottom = Ppos(2) + 10;
pbh = uicontrol(panel,'Style','pushbutton','String','Import',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Fontsize', fontsize,...
  'Fontweight', 'bold',...
  'Tag', 'importBtn');
set(pbh, 'Callback', {'cb_importBtn', pbh, mainfig});
set(pbh, 'TooltipString', 'Retrieve objects to MATLAB workspace');

%-- Save Btn

Bleft   = 120;
Bheight = ALLHEIGHT;
Bwidth  = 100;
Bbottom = Ppos(2) + 10;
pbh = uicontrol(panel,'Style','pushbutton','String','Save',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Fontsize', fontsize,...
  'Fontweight', 'bold',...
  'Tag', 'saveBtn');
set(pbh, 'Callback', {'cb_saveBtn', pbh, mainfig});
set(pbh, 'TooltipString', 'Retrieve objects to disk');


end

%-------------------------------------------------------------------------
% Create the Query panel
%
function panel = create_query_panel(panel, Gproperties)

Ppos    = get(panel, 'Position');
mainfig = findobj('Tag', 'LTPDARepomainfig');

```

```
%-- Build conditions
buildConditions(panel);

end

%-----------------------------------------------------------------------
% Create the submit panel
%
function panel = create_submit_panel(panel, Gproperties)

Ppos    = get(panel, 'Position');
mainfig = findobj('Tag', 'LTPDARepomainfig');
ALLHEIGHT = 25;
%-----------------------------------------------------------------------
%-- Text string
Sleft   = 10;
Sheight = ALLHEIGHT;
Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;
Swidth  = 150;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
sth = uicontrol(panel,'Style','text',...
  'String','LTPDA Objects',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'center',...
  'Position',[Sleft Sbottom Swidth Sheight]);

%-----------------------------------------------------------------------
%-- Workspace list of LTPDA objects
Lleft   = 10;
Lheight = 350;
Lbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Lheight;
Lwidth  = Swidth;

lbh = uicontrol(panel,'Style','listbox',...
  'String',{' '},...
  'Value',1,...
  'BackgroundColor', 'w',...
  'Fontsize', fontsize,...
  'Max', 1000,...
  'Position',[Lleft Lbottom Lwidth Lheight],...
  'Tag', 'workspaceObjsList');

% Set callback
set(lbh, 'Callback', {'cb_workspaceObjsList', lbh, mainfig});

% fill list
objs = getWorkspaceObjs();
setWorkspaceObjsList(objs);

%-----------------------------------------------------------------------
%-- Refresh button
pos     = get(lbh, 'Position');
Bleft   = Lleft;
Bheight = ALLHEIGHT;
Bwidth  = 100;
Bbottom = pos(2) - Bheight - 10;
pbh = uicontrol(panel,'Style','pushbutton','String','Refresh list',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Tag', 'refreshListBtn');
set(pbh, 'Callback', {'cb_refreshObjsList', pbh, mainfig});

%-----------------------------------------------------------------------
%-- Info panel
% pos = get(lbh, 'Position');
% Sleft   = 10 + pos(1) + pos(3);
% Sheight = 150;
% Sbottom = Ppos(2) + Ppos(4) - 40 - Gproperties.Gborder - Sheight;
% Swidth  = 500;
% sth = uicontrol(panel,'Style','text',...
%   'String','',...
%   'Fontsize', fontsize,...
%   'BackgroundColor', 'w',...
%   'ForegroundColor', 'b',...
%   'HorizontalAlignment', 'left',...
%   'Position',[Sleft Sbottom Swidth Sheight],...
%   'Tag', 'ltpdaObjInfo');


%-----------------------------------------------------------------------
%-- experiment title
pos     = get(sth, 'Position');
Sleft   = 20 + pos(1) + pos(3);
Sheight = ALLHEIGHT;
Swidth  = 150;
% Sbottom = pos(2) - 20 - Sheight;
```

```
Sbottom = Ppos(2) + Ppos(4) - 20 - Gproperties.Gborder - Sheight;

sth = uicontrol(panel,'Style','text',...
  'String','Experiment Title',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'FontWeight', 'bold', ...
  'ForegroundColor', 'r', ...
  'Position',[Sleft Sbottom Swidth Sheight]);
Tleft   = Sleft + Swidth + 20;
Theight = ALLHEIGHT;
Twidth  = 380;
Tbottom = Sbottom;

sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'experimentTitle');

ttstr = 'A short title for the experiment from which the data originates.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------------
%-- experiment description
Sheight = ALLHEIGHT;
Tbottom = Tbottom - 20 - ALLHEIGHT;

sth = uicontrol(panel,'Style','text',...
  'String','Experiment description',...
  'Fontsize', fontsize,...
  'FontWeight', 'bold', ...
  'ForegroundColor', 'r', ...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Tbottom Swidth Sheight]);

Extra = 60;
Tbottom = Tbottom + 20 - Theight - Extra;
Theight = ALLHEIGHT + Extra;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 100,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'experimentDescription');

ttstr = 'A description of the experiment from which the data originates.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------------
%-- analysis description
Tbottom = Tbottom - 20 - Sheight;

sth = uicontrol(panel,'Style','text',...
  'String','Analysis description',...
  'Fontsize', fontsize,...
  'FontWeight', 'bold', ...
  'ForegroundColor', 'r', ...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Tbottom Swidth Sheight]);

Extra = ALLHEIGHT;
Tbottom = Tbottom - Extra;
Theight = ALLHEIGHT + Extra;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 100,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'analysisDescription');

ttstr = 'A description of the analysis performed on the data.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------------
%-- quantity
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;
```

```
Theight = ALLHEIGHT + Extra;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 100,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'additionalComments');

ttstr = 'Anything else you want to say about the objects being submitted.';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------
%-- additional authors
Tbottom = Tbottom - Theight;

sth = uicontrol(panel,'Style','text',...
  'String','Additional Authors',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Sleft Tbottom Swidth Sheight]);

Extra = 0;
Tbottom = Tbottom - Extra;
Theight = ALLHEIGHT + Extra;
sth = uicontrol(panel,'Style','edit',...
  'String','',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Max', 100,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'additionalAuthors');

ttstr = 'Who else helped creating these objects?';
set(sth, 'Tooltipstring', ttstr)

%-----------------------------------------------------------------------
%-- Submit Btn

pos     = get(lbh, 'Position');
Bleft   = Lleft;
Bheight = ALLHEIGHT;
Bwidth  = 100;
Bbottom = Ppos(2) + 10;
pbh = uicontrol(panel,'Style','pushbutton','String','Submit',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Fontsize', fontsize,...
  'Fontweight', 'bold',...
  'Tag', 'submitBtn');
set(pbh, 'Callback', {'cb_submitBtn', pbh, mainfig});



end


%-----------------------------------------------------------------------
% Create the server connection panel
%
function panel = create_connection_panel(panel, Gproperties)


% %-- Panel
% Pleft   = Gproperties.Gborder/Gproperties.Gheight;
% Pbottom = Gproperties.Gborder/Gproperties.Gheight;
% Pwidth  = 1-2*Gproperties.Gborder/Gproperties.Gwidth;
% Pheight = 0.4;
%
% hp = uipanel('Parent', mainfig,...
%              'Title', 'Server Connection',...
%              'FontSize', 14,...
%              'BackgroundColor', Gproperties.Gcol,...
%              'Position', [Pleft Pbottom Pwidth Pheight],...
%              'BorderType', 'etchedout', ...
%              'HighlightColor', [0.1 0.9 0.1]);

Ppos    = get(panel, 'Position');
mainfig = findobj('Tag', 'LTPDARepomainfig');

%-- Server selection
Sleft   = 10;
Sbottom = Ppos(2) + Ppos(4) - 40-Gproperties.Gborder;
Swidth  = 150;
```

```
Sheight = 25;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');
servers  = getappdata(0, 'ltpda_repo_servers');

pmh = uicontrol(panel,'Style','popupmenu',...
  'String', servers,...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'Value',1,'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'serverSelect');
set(pmh, 'Callback', {'cb_serverSelect', pmh, mainfig});

%-- Server Txt entry
Tleft   = Sleft + Swidth + 10;
Tbottom = Sbottom;
Twidth  = 200;
Theight = Sheight;

eth = uicontrol(panel,'Style','edit',...
  'String','localhost',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Position',[Tleft Tbottom Twidth Theight],...
  'Tag', 'serverTxt');

%-- Text string
sth = uicontrol(panel,'Style','text',...
  'String','Server hostname',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft+Twidth+10 Tbottom 120 Theight]);

%-- Database selection
pmh = uicontrol(panel,'Style','popupmenu',...
  'String',{'ltpda_test','utp','test'},...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'Value',1,'Position',[Sleft Sbottom-Sheight-10 Swidth Sheight],...
  'Tag', 'dbSelect');
set(pmh, 'Callback', {'cb_dbSelect', pmh, mainfig});

%-- Database Txt entry
eth = uicontrol(panel,'Style','edit',...
  'String','ltpda_test',...
  'Fontsize', fontsize,...
  'BackgroundColor', 'w',...
  'Position',[Tleft Tbottom-Theight-10 Twidth Theight],...
  'Tag', 'databaseTxt');

%-- Text string
sth = uicontrol(panel,'Style','text',...
  'String','Database',...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'HorizontalAlignment', 'left',...
  'Position',[Tleft+Twidth+10 Tbottom-Theight-10 120 Theight]);

%-- Available DBs button
pos     = get(sth, 'Position');
Bleft   = pos(1) + pos(3) + 20;
Bbottom = pos(2);
Bwidth  = 100;
Bheight = Sheight;
pbh = uicontrol(panel,'Style','pushbutton','String','Get DBs',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Tag', 'getDbsBtn');
set(pbh, 'Callback', {'cb_getDBs', pbh, mainfig});

%-- Login Btn
pos     = get(pmh, 'Position');
Bleft   = pos(1);
Bbottom = pos(2) - pos(4) - 20;
Bwidth  = 100;
Bheight = Sheight;

pbh = uicontrol(panel,'Style','pushbutton','String','Connect',...
  'Position',[Bleft Bbottom Bwidth Bheight ],...
  'Tag', 'connectBtn');
set(pbh, 'Callback', {'cb_connectBtn', pbh, mainfig});



end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function sqlResultsGUI(results, fields, query)

% 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 $
%

leftMargin   = 10;
bottomMargin = 100;
topMargin    = 20;

btnHeight = 20;
btnWidth  = 60;
btnGap    = 20;

Color     = [240 240 240]/255;
Height    = 600;
Width     = 800;

Screen    = get(0,'screensize');
Position  = [Screen(3)/2 - Width/2 Screen(4)/2-Height/2 Width Height];

% create figure
```

```
gui.main = figure('HandleVisibility','on',...
    'IntegerHandle','off',...
    'Menubar','none',...
    'NumberTitle','off',...
    'Name','Query Results',...
    'Tag','main',...
    'Color',Color,...
    'Resize', 'on',...
    'Position',Position,...
    'ResizeFcn', {@resizeMain, leftMargin, bottomMargin, topMargin});

% add query string
Tleft   = leftMargin;
Twidth = Width - 2*leftMargin;
Theight = 80;
Tbottom = Position(4) - Theight - 20;
sth = uicontrol(gui.main,'Style','edit',...
                'String',query,...
                'Fontsize', 14,...
                'BackgroundColor','w',...
                'Max', 100,...
                'HorizontalAlignment', 'left',...
                'Tag', 'QueryString', ...
                'Enable', 'on', ...
                'Position',[Tleft Tbottom Twidth Theight], ...
                'Callback', {@qtxtCB, query});

setappdata(gui.main, 'qstring', sth);

if size(fields,2) ~= size(results,2)
  fields = cell(1,size(results,2));
end

% In MATLAB 2008a we can use a proper uitable:
v = ver('MATLAB');
if strcmp(v.Version, '7.6')
  t = uitable(gui.main, 'Data', results, ...
    'ColumnName', fields,...
    'units', 'normalized',...
    'Position', [.01,.08,1-0.02,.73]);
else

  % Otherwise, we use the createTable function which uses a JTable
  [t,b] = createTable(gui.main, fields, results, false, ...
    'AutoResizeMode',javax.swing.JTable.AUTO_RESIZE_OFF,...
    'Editable',false,'Position',[.01,.08,1-0.02,.73]);

end

setappdata(gui.main, 'table', t);

% add export button
btn = uicontrol(gui.main,...
  'Style','pushbutton',...
  'String','Export',...
  'Tag', 'exportBtn',...
  'Callback', {@exportBtn_Callback, gui, results, fields}, ...
  'Position',[leftMargin leftMargin btnWidth btnHeight], 'Visible', 'off');

get(btn)

%-----------------------------------------------------------------------
% Callbacks

% callback to reset the query string in case the user deletes it
function qtxtCB(hObject, eventdata, query)

set(hObject, 'String', query);

%-------------------------------------------------------
% export button
function exportBtn_Callback(hObject, eventdata, gui, results, fields)

% get table
t   = getappdata(gui.main, 'table');
table = get(t.Table);

rows = table.SelectedRows + 1;
cols = table.SelectedColumns + 1;
res = results(rows, cols);
nResults = size(res, 1);

% get file to export to
[filename, pathname, filteridx] = uiputfile('*.txt', 'Select export file');

fd = fopen(filename, 'w+');
```

```
% header
fprintf(fd, '# ');
fprintf(fd, '%s\t|\t', fields{cols});
fprintf(fd, '\n');


for j=1:nResults
  for k=1:size(res,2)
    r = res{j,k};
    if ischar(r)
      fprintf(fd, '%s\t\t', char(r));
    elseif isnumeric(r)
      fprintf(fd, '%f\t\t', r);
    else
      error('### Unknown data format.');
    end
  end
  fprintf(fd, '\n');
end
fclose(fd);

edit(filename);

%-------------------------------------------------------
% Resize Main figure
function resizeMain(hObject, eventdata, leftMargin, bottomMargin, topMargin)

% resize table
s   = getappdata(hObject, 'qstring');
ps  = get(s, 'Position');
t   = getappdata(hObject, 'table');
fs  = get(hObject, 'Position');
pos = get(t, 'Position');

% set(t, 'Position', [pos(1) pos(2) fs(3)-2*leftMargin fs(4)-bottomMargin-ps(4)]);
set(s, 'Position', [ps(1) fs(4)-ps(4)-20 fs(3)-2*leftMargin ps(4)]);


% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

- repogui 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 ^

```
function hh = uitabpanel(varargin)

% 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 $
%
%


% Author: Shiying Zhao (zhao@arch.umsl.edu)
% Version: 1.0
% First created: May 02, 2006
% Last modified: May 20, 2006

tag = '';
units = 'normalized';
position = [0,0,1,1];
fontweight = 'normal';
[style,type] = deal('lefttop',2);
visible = 'on';

foregroundcolor = [1,1,1];
backgroundcolor = [0.7725,0.8431,0.9608];
panelbackgroundcolor = [0.4314,0.5882,0.8431];
titlehighlightcolor = [1,0.8,0];

```

```
horizontalalignment = 'center';
[createfcn,deletefcn,resizefcn] = deal({},{},{});

active = 0;
styles = {'popup','lefttop','righttop','leftbottom','rightbottom', ...
  'centertop','centerbottom'};

for k=1:2:nargin
  property = lower(varargin{k});
  value = varargin{k+1};
  switch property
    case {'style','tabposition','titleposition'}
      style = value;
      type = strmatch(lower(style),styles);
      if isempty(type)
        error(['Bad value for Style/TitlePosition property: ' style '.']);
      end
    case 'fontsize'
      fontsize = value;
    case {'title','string'}
      string = value;
      ntab = length(string);
    case {'frameposition','position'}
      position = value;
    case {'framebackgroundcolor','backgroundcolor'}
      backgroundcolor = value;
    case {'framebordertype','bordertype'}
      bordertype = value;
    case {'panelheights','heights'}
      panelheight = value;
    case {'titleforegroundcolor','foregroundcolor'}
      foregroundcolor = value;
    case {'selecteditem','active'}
      active = value;
    otherwise
      eval([property '= value;']);
  end
end

if ~exist('parent','var')
  parent = gcf;
end
if ~exist('margins','var')
  if type==1
    margins = {[1.5,0.75,1.5,1],'characters'};
  else
    margins = {1,'pixels'};
  end
end
if ~iscell(margins)
  margins = {margins,units};
end

if ~exist('bordertype','var')
  if type==1
    bordertype = 'etchedin';
  else
    bordertype = 'none';
  end
end

if ~exist('titlebackgroundcolor','var')
  titlebackgroundcolor = 0.95*panelbackgroundcolor;
end

cmenu = uicontextmenu;
uimenu(cmenu,'Label','Goto Tab');
uimenu(cmenu,'Separator','on');

htab = uipanel( ...
  'Parent',parent, ...
  'Units',units, ...
  'Position',position, ...
  'DeleteFcn',deletefcn, ...
  'ResizeFcn',resizefcn, ...
  'BackgroundColor',backgroundcolor, ...
  'BorderType',bordertype, ...
  'UIContextMenu',cmenu, ...
  'Visible',visible, ...
  'FontSize', fontsize,...
  'Tag',tag);
set(htab,'Units','pixels');
tabpos = get(htab,'Position');

status = uicontainer( ...
  'Parent',htab, ...
```

```
  'BackgroundColor',backgroundcolor, ...
  'Units',margins{2}, ...
  'Position',[1,1,1,1], ...
  'UIContextMenu',cmenu, ...
  'Visible','off');
set(status,'Units','pixels');
charsz = get(status,'Position');
margins = [charsz(1:2),charsz(1:2)].*margins{1};
set(status,'Units','characters');
set(status,'Position',[1,1,1,1]);
set(status,'Units','pixels');
charsz = get(status,'Position');
titleheight = 2*charsz(4);

if type==1
  %-----------------------------------------------------------------------------
  %  PopupTab
  %-----------------------------------------------------------------------------
  panelpos = tabpos;
  titlepos = [margins(1),tabpos(2)+tabpos(4)-titleheight-margins(2), ...
    tabpos(3)-sum(margins([1,3])),titleheight];
  ttextpos = [0.05,0.1,0.9,0.76];

  if ~exist('panelheight','var')
    panelheight = ones(1,ntab);
  end
  if ~exist('panelbordertype','var')
    panelbordertype = 'line';
  end

  for k=1:ntab
    title(k) = uicontrol( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'Position',titlepos, ...
      'ForegroundColor',foregroundcolor, ...
      'BackgroundColor',titlebackgroundcolor, ...
      'FontWeight',fontweight, ...
      'FontSize', fontsize,...
      'HorizontalAlignment',horizontalalignment, ...
      'Callback',{@PopupTabCbk,k}, ...
      'Style','toggle', ...
      'String',string{k}, ...
      'Value',0, ...
      'Visible','on');

    panel(k) = uipanel( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'Position',titlepos+[0,0,0,panelheight(k)*charsz(4)], ...
      'BackgroundColor',panelbackgroundcolor, ...
      'BorderType',panelbordertype, ...
      'Visible','off');

    uimenu(cmenu, ...
      'Label',string{k}, ...
      'Callback',{@PopupTabCbk,k});
  end

  uimenu(cmenu,'Separator','on');
  uimenu(cmenu, ...
    'Label','Expand All', ...
    'Callback',{@PopupTabEnableAllCbk,'on'});
  uimenu(cmenu, ...
    'Label','Collapse All', ...
    'Callback',{@PopupTabEnableAllCbk,'off'});

  sliderpos = [tabpos(3)-14,2,15,tabpos(4)-3];
  slider = uicontrol( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'Position',sliderpos, ...
    'BackgroundColor',panelbackgroundcolor, ...
    'Style','slider', ...
    'Callback',@PopupTabSliderCbk, ...
    'Value',0, ...
    'Min',-1, ...
    'Max',0, ...
    'Visible','off');

  statuspos = titlepos;
  statuspos([2,4]) = [margins(4),panelpos(4)-(ntab+1)*(margins(4)+titlepos(4))];
  set(status,'Position',statuspos);
  PopupTabCbk(title(1),[],active);
  set(status,'ResizeFcn',@PopupTabResizeCbk);

```

```
else
  %-------------------------------------------------------------------------
  %  TopT/Bottom Tab
  %-------------------------------------------------------------------------
  margins(3:4) = -(margins(1:2)+margins(3:4));
  switch type
    case 2 %lefttop
      [loop,sign] = deal(1:ntab,[1,0,1,0]);
    case 3 %righttop
      [loop,sign] = deal(ntab:-1:1,[ntab,0,0,0]);
    case 4 %lefttbottom
      [loop,sign] = deal(1:ntab,[1,0,1,1]);
    case 5 %rightbottom
      [loop,sign] = deal(ntab:-1:1,[ntab,0,0,1]);
    case 6 %centertop
      [loop,sign] = deal(1:ntab,[1,ntab,1,0]);
    case 7 %centerbottom
      [loop,sign] = deal(1:ntab,[1,ntab,1,1]);
  end
  addtitle = 2*[sign(3:4),-1,-1];
  if sign(3)
    ttextpos = [0.05,0.08,0.9,0.76];
  else
    ttextpos = [0.05,0.05,0.9,0.76];
  end

  if ~exist('panelbordertype','var')
    panelbordertype = 'beveledout';
  end

  for k=1:ntab
    title(k) = uipanel( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'BackgroundColor',titlebackgroundcolor, ...
      'BorderType',panelbordertype, ...
      'ButtonDownFcn',{@TopBottomTabCbk,k}, ...
      'Visible','on');
    %titlewidth(k) = (length(string{k})+4)*1.2*charsz(3);

    ttext(k) = uicontrol( ...
      'Parent',title(k), ...
      'Units','normalized', ...
      'Position',ttextpos, ...
      'ForegroundColor',foregroundcolor, ...
      'BackgroundColor',titlebackgroundcolor, ...
      'HorizontalAlignment',horizontalalignment, ...
      'Style','text', ...
      'String',string{k}, ...
      'Enable','inactive', ...
      'FontSize', fontsize,...
      'HitTest','off',...
      'Visible','on');

    panel(k) = uipanel( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'BackgroundColor',panelbackgroundcolor, ...
      'BorderType',panelbordertype, ...
      'Visible','off');

    uimenu(cmenu, ...
      'Label',string{k}, ...
      'Callback',{@TopBottomTabCbk,k});
  end

  if ~active, active = 1; end
  set(panel(active),'Visible','on');

  inset(1) = uicontrol( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'BackgroundColor',panelbackgroundcolor, ...
    'Style','text', ...
    'Visible','on');
  if ~isnan(titlehighlightcolor)
    inset(2) = uicontrol( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'BackgroundColor',titlehighlightcolor, ...
      'Style','text', ...
      'Visible','on');
    inset(3) = uicontrol( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'BackgroundColor',0.85*titlehighlightcolor, ...
```

```
      'Style','text', ...
      'Visible','on');
  end

  TopBottomTabResizeCbk(status,[]);
  set(status,'ResizeFcn',@TopBottomTabResizeCbk);
end

setappdata(htab,'panels',panel);
setappdata(htab,'status',status);
if ~isempty(createfcn)
  createfcn(htab,[],panel,status);
end

if nargout, hh = htab; end


%-----------------------------------------------------------------------
% Inner Functions for Popup Tab
%-----------------------------------------------------------------------
  function PopupTabCbk(hobj,evdt,indx)

    set(title,'Value',0);
    if indx>0
      indxvis = strcmp(get(panel(indx),'Visible'),'off');
      if indxvis
        set(panel(indx),'Visible','on');
      else
        set(panel(indx),'Visible','off');
      end
    end
    if isempty(strmatch('on',get(panel,'Visible')))
      set(status,'Visible','on');
    else
      set(status,'Visible','off');
    end

    currval = get(slider,'Value');
    if currval<0
      set(slider,'Value',0);
      PopupTabSliderCbk(slider,evdt);
    end
    titlepos = get(title(1),'Position');
    addtolen = titlepos(2)+titlepos(4)+margins(4);

    for k=1:ntab
      titlepos = get(title(k),'Position');
      titlepos(2) = addtolen-titlepos(4)-margins(4);
      set(title(k),'Position',titlepos);
      addtolen = titlepos(2);
      visible = strcmp(get(panel(k),'Visible'),'on');
      if visible
        panelpos = get(panel(k),'Position');
        addtolen = addtolen-panelpos(4);
        panelpos(2) = addtolen;
        set(panel(k),'Position',panelpos);
      end
    end

    if visible
      botpos = get(panel(ntab),'Position');
    else
      botpos = get(title(ntab),'Position');
    end

    visible = strcmp(get(slider,'Visible'),'on');
    addwidth = (-1)^(~visible);
    if botpos(2)*addwidth>0
      if visible
        set(slider,'Visible','off');
      else
        set(slider,'Visible','on');
      end
    else
      if visible
        addwidth = 0;
      else
        return;
      end
    end

    addwidth = addwidth*sliderpos(3);
    for k=1:ntab
      titlepos = get(title(k),'Position');
      titlepos(3) = titlepos(3)+addwidth;
      set(title(k),'Position',titlepos);
```

```
      set(title(k),'UserData',titlepos(2));

      panelpos = get(panel(k),'Position');
      panelpos(3) = panelpos(3)+addwidth;
      set(panel(k),'Position',panelpos);
      set(panel(k),'UserData',panelpos(2));
    end

    toplen = botpos(2)-margins(2);
    if toplen>0
      set(slider,'Value',0);
      return;
    end

    try
      if indxvis
        viewpos = get(panel(indx),'Position');
        if viewpos(2)-viewpos(4)<0
          if viewpos(4)+margins(4)<=tabpos(4);
            currval = min(0,viewpos(2)-margins(4)/2);
          else
            viewpos = get(title(indx),'Position');
            currval = viewpos(2)+viewpos(4)+margins(4)/2-tabpos(4);
          end
        end
      end
    catch
    end

    set(slider,'Min',toplen);
    set(slider,'Value',max(currval,toplen));
    set(slider,'SliderStep',[0.3,1]/max(1,-toplen/100));
    PopupTabSliderCbk(slider,evdt);

  end


%-----------------------------------------------------------------------
  function PopupTabSliderCbk(hobj,evdt)

    currval = get(hobj,'Value');
    for k=1:ntab
      titlepos = get(title(k),'Position');
      titlepos(2) = get(title(k),'UserData')-currval;
      set(title(k),'Position',titlepos);
      panelpos = get(panel(k),'Position');
      panelpos(2) = get(panel(k),'UserData')-currval;
      set(panel(k),'Position',panelpos);
    end

  end


%-----------------------------------------------------------------------
  function PopupTabEnableAllCbk(hobj,evdt,cstr)

    set(panel,'Visible',cstr);
    PopupTabCbk(hobj,evdt,0);

  end


%-----------------------------------------------------------------------
  function PopupTabResizeCbk(hobj,evdt)

    oldpos = tabpos;
    units = get(htab,'Units');
    set(htab,'Units','pixels');
    tabpos = get(htab,'Position');
    set(htab,'Units',units);

    addtopos = tabpos-oldpos;
    sliderpos([1,2,4]) = sliderpos([1,2,4])+addtopos([1,2,4]);
    set(slider,'Position',sliderpos);

    addtopos = [tabpos(3:4)-oldpos(3:4),0,0];
    currval = get(slider,'Value');
    if currval<0
      set(slider,'Value',0);
      PopupTabSliderCbk(slider,evdt);
    end

    for k=1:ntab
      titlepos = get(title(k),'Position');
      set(title(k),'Position',titlepos+addtopos);
    end
```

```
    PopupTabCbk(title(1),evdt,0);

    visible = strcmp(get(slider,'Visible'),'on');
    if visible & currval<0
      set(slider,'Value',max(currval,get(slider,'Min')));
      PopupTabSliderCbk(slider,evdt);
    end

  end


%-----------------------------------------------------------------------
% Inner Functions for Top/Bottom Tab
%-----------------------------------------------------------------------
  function TopBottomTabCbk(hobj,evdt,indx)

    if active==indx, return; end

    % deactivate tab(active)
    n = active;
    titlepos = get(title(n),'Position');
    titlepos = titlepos+addtitle.*[n==sign(1),1,n==sign(1),1];
    titlepos(3) = titlepos(3)+addtitle(3)*(n==sign(2));
    set(title(n),'Position',titlepos,'BackgroundColor',titlebackgroundcolor);
    set(ttext(n),'FontWeight','normal','BackgroundColor',titlebackgroundcolor);
    set(panel(n),'Visible','off');

    % activate tab(indx)
    active = indx;
    titlepos = get(title(indx),'Position');
    titlepos = titlepos-addtitle.*[indx==sign(1),1,indx==sign(1),1];
    titlepos(3) = titlepos(3)-addtitle(3)*(indx==sign(2));

    set(title(indx),'Position',titlepos,'BackgroundColor',panelbackgroundcolor);
    set(ttext(indx),'FontWeight','bold','BackgroundColor',panelbackgroundcolor);
    insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
    set(inset(1),'Position',insetpos);
    try
      insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
      set(inset(2),'Position',insetpos);
      set(inset(3),'Position',insetpos+[2,1-2*sign(4),-4,0]);
    catch
    end
    set(panel(indx),'Visible','on');

    if isappdata(htab,'SelectionChangeFcn')
      SelectionChangeFcn = getappdata(htab,'SelectionChangeFcn');
      eventdata = struct('EventName','SelectionChanged', ...
        'OldValue',n,'NewValue',indx,'Panels',panel,'Status',status);
      if iscell(SelectionChangeFcn)
        SelectionChangeFcn{1}(hobj,eventdata,SelectionChangeFcn{2:end});
      else
        SelectionChangeFcn(hobj,eventdata);
      end
    end

  end


%-----------------------------------------------------------------------
  function TopBottomTabResizeCbk(hobj,evdt)

    units = get(htab,'Units');
    set(htab,'Units','pixels');
    tabpos = get(htab,'Position');
    set(htab,'Units',units);

    titlepos = [0,0,0,titleheight];
    if sign(4)
      panelpos = [0,titleheight+1,tabpos(3),tabpos(4)-titleheight]+margins;
      titlepos(2) = panelpos(2)-titlepos(4);
    else
      panelpos = [0,0,tabpos(3),tabpos(4)-titleheight]+margins;
      titlepos(2) = panelpos(2)+panelpos(4);
    end
    set(panel,'Position',panelpos);

    n = sum(cellfun('length',string))+2*ntab;
    if type<6
      if (n+2*ntab)*1.2*charsz(3)+2>panelpos(3)
        [loop,sign(1:3)] = deal(1:ntab,[1,ntab,1]);
      else
        if type==2 | type==4
          [loop,sign(1:3)] = deal(1:ntab,[1,0,1]);
        else
          [loop,sign(1:3)] = deal(ntab:-1:1,[ntab,0,0]);
```

```
        end
      end
      addtitle = 2*[sign(3:4),-1,-1];
    end
    titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
    for k=loop
      if sign(2)
        titlepos(3) = panelpos(3)*(length(string{k})+2)/n;
        addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
      else
        titlepos(3) = (length(string{k})+4)*1.2*charsz(3);
        addtopos = addtitle.*[k==sign(1),1,k==sign(1),1];
      end
      titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
      set(title(k),'Position',titlepos+addtopos);
      titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
    end

    statuspos = titlepos;
    statuspos(1) = sign(3)*titlepos(1)+2;
    statuspos(3) = titlepos(1)+sign(3)*(tabpos(3)-2*titlepos(1))-4;

    titlepos = get(title(active),'Position');
    titlepos = titlepos-addtitle.*[active==sign(1),1,active==sign(1),1];
    titlepos(3) = titlepos(3)-addtitle(3)*(active==sign(2));
    set(title(active),'Position',titlepos,'BackgroundColor',panelbackgroundcolor);
    set(ttext(active),'FontWeight','bold','BackgroundColor',panelbackgroundcolor);

    insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
    set(inset(1),'Position',insetpos);
    try
      insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
      set(inset(2),'Position',insetpos);
      set(inset(3),'Position',insetpos+[2,1-2*sign(4),-4,0]);
    catch
    end
    try
      % This causes the above statements being excuted twice for each resizing.
      set(status,'Position',statuspos,'Visible','on');
    catch
      set(status,'Visible','off');
    end

  end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
- 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
- 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
- Callback to refresh the list of objects.
   <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsListQL">m/gui/ltpdaRepoGUI/callbacks/cb_refreshObjsLi
- 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_workspaceObjsL
- Callback to fill the list of workspace objects
   <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObjsListQL">m/gui/ltpdaRepoGUI/callbacks/cb_workspaceObj
- Callback to fill the list of workspace objects
   <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/fillClassMethodsList">m/gui/ltpdaRepoGUI/callbacks/fillClassMethodsL
- 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/setWorkspaceObjsL
- Fill the workspace object list
   <a href="matlab:help
m/gui/ltpdaRepoGUI/callbacks/setWorkspaceObjsListQL">m/gui/ltpdaRepoGUI/callbacks/setWorkspaceObj
- Fill the workspace object list
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function buildConditions(varargin)

% Callback to build the query conditions panel.
%
% $Id: buildConditions.m,v 1.5 2008/02/23 14:24:55 hewitson Exp $
%

ALLHEIGHT = 25;
fontsize = 10;

panel   = varargin{1};
Ppos    = get(panel, 'Position');
mainfig = findobj('Tag', 'LTPDARepomainfig');
Nc      = getappdata(mainfig, 'Nconditions');
conn    = getappdata(mainfig, 'connection');
Gproperties = getappdata(mainfig, 'Gproperties');

% before deleting, record the table selection
tableSelected = getappdata(mainfig, 'tableSelected');
% record field selection
fieldList = findobj('Tag', 'fieldsList');
fselection = get(fieldList, 'Value');
% record order by
fieldOrder = findobj('Tag', 'fieldOrderBy');
forder     = get(fieldOrder, 'Value');
% record sort direction
sortDir    = findobj('Tag', 'sortDir');
sort       = get(sortDir, 'Value');
% record current conditions
conds = cell(Nc,4);
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;
```

```
Sbottom = Ppos(2) + 60; %listpos(2)  - 40 - Sheight;
Swidth  = 60;

sth = uicontrol(panel,'Style','text',...
                'String', 'Query:',...
                'Fontsize', fontsize,...
                'Max', 5,...
                'BackgroundColor', Gproperties.Gcol,...
                'HorizontalAlignment', 'left',...
                'Position', [Sleft Sbottom Swidth Sheight]);

Sleft   = 10 + Swidth + 20;
Swidth  = 650;
Sheight = 100;
Sbottom = Ppos(2) + 60; %listpos(2)  - 40 - Sheight;
sth = uicontrol(panel,'Style','edit',...
                'String', '',...
                'Fontsize', fontsize,...
                'Max', 5,...
                'BackgroundColor', 'w',...
                'HorizontalAlignment', 'left',...
                'Position', [Sleft Sbottom Swidth Sheight],...
                'Tag', 'queryDisplayTxt');



%-----------------------------------------------------------------------
%-- Draw conditions

conditions = {'=', '<>', '<=', '>=', '<', '>', 'like', 'not like', 'is'};
compounds  = {'AND', 'OR', 'XOR'};

pos = get(lbh, 'Position');
Sleft   = pos(1) + pos(3) + 20;
Sheight = ALLHEIGHT;
Sbottom = Ppos(2) + Ppos(4) - 25 - Sheight;
Swidth  = 50;

sth = uicontrol(panel,'Style','text',...
                'String', 'Where',...
                'Fontsize', fontsize,...
                'BackgroundColor', Gproperties.Gcol,...
                'HorizontalAlignment', 'left',...
                'Position',[Sleft Sbottom Swidth Sheight]);


% pos = get(sth, 'Position');
Cheight = ALLHEIGHT;
STEP    = 5;
for k=1:Nc
  Cleft   = pos(1) + pos(3) + 10 + STEP;
  Cbottom = pos(2) + pos(4) -25 - (k-1)*(Cheight+10);
  Cwidth  = 150;
  % Field Dropdown
  pmh = uicontrol(panel,'Style','popupmenu',...
    'String', fields,...
    'Fontsize', fontsize,...
    'BackgroundColor', 'w', ...
    'Value',1,'Position',[Cleft Cbottom Cwidth Cheight],...
    'Tag', sprintf('c%02dField', k));
  set(pmh, 'Callback', 'buildquery');
  if ~isempty(conds{k,1})
    set(pmh, 'Value', conds{k,1});
  end
%   Gproperties.Gcol,
  % Condition Dropdown
  Cleft  = Cleft + Cwidth + STEP;
  Cwidth = 100;
  pmh = uicontrol(panel,'Style','popupmenu',...
    'String', conditions,...
    'Fontsize', fontsize,...
    'BackgroundColor', 'w',...
    'Value',1,'Position',[Cleft Cbottom Cwidth Cheight],...
    'Tag', sprintf('c%02dCondition', k));
  set(pmh, 'Callback', 'buildquery');
  if ~isempty(conds{k,2})
    set(pmh, 'Value', conds{k,2});
  end

  % Value txt box
  Cleft  = Cleft + Cwidth + STEP;
  Cwidth = 150;
  sth = uicontrol(panel,'Style','edit',...
    'String','',...
    'Fontsize', fontsize,...
    'BackgroundColor', 'w',...
```

```
    'HorizontalAlignment', 'left',...
    'Position',[Cleft Cbottom Cwidth Cheight],...
    'Tag', sprintf('c%02dValue', k));
  set(sth, 'Callback', 'buildquery');
  if ~isempty(conds{k,3})
    set(sth, 'String', conds{k,3});
  end

  % Add compound statement
  if k<Nc
    Cleft  = Cleft + Cwidth + STEP;
    Cwidth = 80;
    pmh = uicontrol(panel,'Style','popupmenu',...
      'String', compounds,...
      'Fontsize', 10,...
      'BackgroundColor', Gproperties.Gcol,...
      'Value',1,'Position',[Cleft Cbottom Cwidth Cheight],...
      'Tag', sprintf('c%02dCompound', k));
    set(pmh, 'Callback', 'buildquery');
    if ~isempty(conds{k,4})
      set(pmh, 'Value', conds{k,4});
    end
  end

end

%-----------------------------------------------------------------------------
%-- add +/- buttons
k = Nc+1;
Bbottom = pos(2) + pos(4)  -25 - (k-1)*(Cheight+10);
Bleft   = pos(1) + pos(3) + 20;
Bheight = ALLHEIGHT;
Bwidth  = 20;
pbh = uicontrol(panel,'Style','pushbutton','Fontsize', fontsize, 'String','+',...
                'Position',[Bleft Bbottom Bwidth Bheight ],...
                'Tag', 'condPlusBtn');
set(pbh, 'Callback', {'cb_condPlusBtn', pbh, panel, mainfig});
Bleft   = pos(1) + pos(3) + 20 + 30;
pbh = uicontrol(panel,'Style','pushbutton','Fontsize', fontsize, 'String','-',...
                'Position',[Bleft Bbottom Bwidth Bheight ],...
                'Tag', 'condSubtractBtn');
set(pbh, 'Callback', {'cb_condSubtractBtn', pbh, panel, mainfig});


%-----------------------------------------------------------------------------
%-- order by
k = Nc + 2;
Sleft   = pos(1) + pos(3) + 20;
Sheight = ALLHEIGHT;
Swidth  = 60;
Sbottom = pos(2) + pos(4)  - 25 -(k-1)*(Cheight+10);
sth = uicontrol(panel,'Style','text',...
                'String', 'Order by',...
                'Fontsize', fontsize,...
                'BackgroundColor', Gproperties.Gcol,...
                'HorizontalAlignment', 'left',...
                'Position',[Sleft Sbottom Swidth Sheight]);
% Gproperties.Gcol
% Field Dropdown
Sleft = pos(1) + pos(3) + 20 + Swidth + 20;
Swidth = 100;
pmh = uicontrol(panel,'Style','popupmenu',...
  'String', fields,...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'Value',1,'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'fieldOrderBy');
set(pmh, 'Callback', 'buildquery');
if ~isempty(forder)
  set(pmh, 'Value', forder);
end

% Desc Dropdown
Sleft = Sleft + 20 + Swidth;
Swidth = 100;
pmh = uicontrol(panel,'Style','popupmenu',...
  'String', {'DESC', 'ASC'},...
  'Fontsize', fontsize,...
  'BackgroundColor', Gproperties.Gcol,...
  'Value',1,'Position',[Sleft Sbottom Swidth Sheight],...
  'Tag', 'sortDir');
set(pmh, 'Callback', 'buildquery');
if ~isempty(sort)
  set(pmh, 'Value', sort);
end

```

```
%-------------------------------------------------------------------------
%-- Run query

k = Nc + 3;
Sleft   = Ppos(1) + 10;
Sheight = ALLHEIGHT;
Swidth  = 100;
Sbottom = Ppos(2) + 10;
pbh = uicontrol(panel,'Style','pushbutton','String','Execute query',...
                 'Position',[Sleft Sbottom Swidth Sheight ],...
                 'Tag', 'refreshListBtn');
set(pbh, 'Callback', {'cb_executeQuery', pbh, mainfig});


% Build query
buildquery()

drawnow

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function buildquery()

% 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 $
%

mainfig    = findobj('Tag', 'LTPDARepomainfig');

% Get number of conditions
Nc  = getappdata(mainfig, 'Nconditions');

% Select fields
fieldList = findobj('Tag', 'fieldsList');
fieldsStr = get(fieldList, 'String');
vals      = get(fieldList, 'Value');
fields    = fieldsStr(vals);

% Get table
tableSel = findobj('Tag', 'tableSelect');
tbls     = get(tableSel, 'String');
tbl      = tbls{get(tableSel, 'Value')};

% Start the query
q = 'SELECT ';
for j=1:length(fields)
  q = [q fields{j} ','];
end
q = [q(1:end-1) ' FROM ' tbl];


for k=1:Nc

  if k==1
```

```
    q = [q ' WHERE '];
  end

  % get field
  h = findobj('Tag', sprintf('c%02dField', k));
  fields = get(h, 'String');
  field  = fields{get(h, 'Value')};

  q = [q field];

  % get condition
  h = findobj('Tag', sprintf('c%02dCondition', k));
  conds = get(h, 'String');
  cond  = conds{get(h, 'Value')};

  q = [q ' ' cond ' '];

  % get value
  h   = findobj('Tag', sprintf('c%02dValue', k));
  val = get(h, 'String');
  q = [q '"' val '"'];

  % Get compound
  if k<Nc

    h   = findobj('Tag', sprintf('c%02dCompound', k));
    comps = get(h, 'String');
    comp  = comps{get(h, 'Value')};

    q = [q ' ' comp ' '];

  end


end

% Order by
h = findobj('Tag', 'fieldOrderBy');
fields = get(h, 'String');
field  = fields{get(h, 'Value')};

q = [q ' ORDER BY ' field];

h = findobj('Tag', 'sortDir');
dirs = get(h, 'String');
dir  = dirs{get(h, 'Value')};

q = [q ' ' dir ';'];

% Set query string on GUI
h = findobj('Tag', 'queryDisplayTxt');
set(h, 'String', q);

drawnow
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_classMethodList(varargin)

% 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 $
%

myh     = varargin{1};
mainfig = varargin{4};

id = findobj('Tag', 'methodInfoPanel');

% get selected method
id = findobj('Tag', 'classMethodList');
ostr = get(id, 'String');
oidx = get(id, 'Value');

objStr = ostr(oidx);

id = findobj('Tag', 'methodInfoPanel');

if length(oidx) == 1
  s = help(char(objStr));
  set(id, 'String', s);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_condPlusBtn(varargin)

% Add a condition to the query panel
%
% M Hewitson
%
% $Id: cb_condPlusBtn.m,v 1.3 2008/02/23 13:44:08 hewitson Exp $
%

myh = varargin{1};

mainfig = findobj('Tag', 'LTPDARepomainfig');
Nc = getappdata(mainfig, 'Nconditions');
if Nc >= 8
  Nc = 7;
  warning('!!! Can''t have more than 8 conditions.');
end

setappdata(mainfig, 'Nconditions', Nc + 1);
buildConditions(get(myh, 'Parent'));


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_condSubtractBtn(varargin)

% Remove a condition from the query panel
%
% M Hewitson
%
% $Id: cb_condSubtractBtn.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
%

myh = varargin{1};

mainfig = findobj('Tag', 'LTPDARepomainfig');
Nc = getappdata(mainfig, 'Nconditions');
if Nc < 1
  Nc = 1;
end
setappdata(mainfig, 'Nconditions', Nc - 1);

buildConditions(get(myh, 'Parent'));


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_connectBtn(varargin)

% Callback to connect to server
%
% M Hewitson
%
% $Id: cb_connectBtn.m,v 1.5 2008/08/11 15:00:06 hewitson Exp $
%

myh     = varargin{1};
mainfig = varargin{4};
status  = findobj('Tag', 'statusTxt');
panels = getappdata(mainfig, 'panels');

% Get conn
conn = getappdata(mainfig, 'connection');

if isempty(conn)

  % get hostname
  hostname = get(findobj('Tag','serverTxt'), 'String');

  % get the database
  db = get(findobj('Tag','databaseTxt'), 'String');

  % check for username and password
  dbuser = getappdata(mainfig, 'dbuser');
  dbpass = getappdata(mainfig, 'dbpass');

  % connect
  if ~isempty(dbpass) && ~isempty(dbuser)
    conn = utils.mysql.connect(hostname, db, dbuser, dbpass);
  else
    [conn, dbpass] = utils.mysql.connect(hostname, db);
  end
  if isa(conn, 'database')
    setappdata(mainfig, 'connection', conn);
    setappdata(mainfig, 'username', conn.Username);
    set(myh, 'String', 'disconnect');
```

```
    set(status, 'String', sprintf('connected to %s on %s as %s', db, hostname, conn.Username));

    % Set tables on query panel
    buildConditions(panels(3));

    % store the connection
    setappdata(mainfig, 'dbuser', conn.Username);
    setappdata(mainfig, 'dbpass', dbpass);
  else
    set(status, 'String', 'not connected');
    error('Invalid connection.');
  end
else
  % disconnect
  close(conn);
  conn = [];
  disp('* disconnected.');
  setappdata(mainfig, 'connection', conn);
  setappdata(mainfig, 'username', '');
  set(myh, 'String', 'connect');
  set(status, 'String', 'not connected');
  setappdata(mainfig, 'dbuser', '');
  setappdata(mainfig, 'dbpass', '');
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_dbSelect(varargin)

% Callback for when user selects a database
%
% M Hewitson
%
% $Id: cb_dbSelect.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
%

myh        = varargin{1};
mainfig    = findobj('Tag', 'LTPDARepomainfig');

% Get selected DB
dbh = findobj('Tag', 'dbSelect');
dbs = get(dbh, 'String');
db  = dbs{get(dbh, 'Value')};

% set selected DB
dbh = findobj('Tag', 'databaseTxt');
set(dbh, 'String', db);


end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_executeQuery(varargin)

% 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 $
%

% Get connection
mainfig  = findobj('Tag', 'LTPDARepomainfig');
conn     = getappdata(mainfig, 'connection');
if isempty(conn)
  error('### Please connect to a database first.');
end

% Get Selected fields
fieldList = findobj('Tag', 'fieldsList');
fieldsStr = get(fieldList, 'String');
vals      = get(fieldList, 'Value');
fields    = fieldsStr(vals)';


% Get query
h = findobj('Tag', 'queryDisplayTxt');
q = get(h, 'String');

try
  curs = exec(conn, q);
  curs = fetch(curs);
  results = curs.Data;
  close(curs);
catch
  warning('!!! Unable to execute query %s', q)
  warning('!!! Server returned: %s', curs.Message);
  results = {};
end

if ~isempty(results)
  if strcmp(results{1}, 'No Data')
    errordlg('Query returned no results.', 'Query Empty');
    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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_fieldSelect(varargin)

% Callback executed when a field is selected
%
% M Hewitson
%
% $Id: cb_fieldSelect.m,v 1.3 2008/02/07 19:27:37 hewitson Exp $
%

%---- Set the order by field to the first in this list

% Get selected fields
fieldList = findobj('Tag', 'fieldsList');
fieldsStr = get(fieldList, 'String');
vals      = get(fieldList, 'Value');
fields    = fieldsStr(vals);

orderBy = findobj('Tag', 'fieldOrderBy');
set(orderBy, 'Value', 1);
set(orderBy, 'String', fields);


% Build query
buildquery();

```

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 ^

```
function cb_getDBs(varargin)

% 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 $
%

myh     = varargin{1};
mainfig = varargin{4};

tables = {'collections', 'objmeta', 'objs', 'transactions', 'users'};

% Get conn
conn = getappdata(mainfig, 'connection');
% Assume I don't need to open the connection
wasEmpty = 0;

% Open connection, if necessary
if isempty(conn)

  % we opened the connection
  wasEmpty = 1;

  % have we a stored username and password
  dbuser = getappdata(mainfig, 'dbuser');
  dbpass = getappdata(mainfig, 'dbpass');

  % get hostname
  hostname = get(findobj('Tag','serverTxt'), 'String');

  % connect
  if ~isempty(dbuser) && ~isempty(dbpass)
    conn = mysql_connect(hostname, 'test', dbuser, dbpass);
  else
    [conn, dbpass] = mysql_connect(hostname);
  end
  if ~isa(conn, 'database')
    error('# couldn''t connect to server');
  end

```

```
  % store the connection
  setappdata(mainfig, 'dbuser', conn.Username);
  setappdata(mainfig, 'dbpass', dbpass);

%
%   % update the connect button and status string
%   status  = findobj('Tag', 'statusTxt');
%   connBtn = findobj('Tag', 'connectBtn');
%   set(connBtn, 'String', 'disconnect');
%   set(status, 'String', sprintf('connected to %s on %s as %s', db, hostname, conn.Username));

end

% get the databases
q = 'show databases';
curs = exec(conn, q);
curs = fetch(curs);

ltpda_dbs = [];
% Now get the LTPDA databases
for j=1:length(curs.Data)
  db = curs.Data{j};

  if ~strcmp(db, 'information_schema')

    q = sprintf('show tables from %s', db);
    c = exec(conn, q);
    c = fetch(c);
    match = 0;
    for l=1:length(c.Data)
      for k=1:length(tables)
        if strcmp(c.Data{l}, tables{k})
          match = match + 1;
        end
      end
    end
    close(c);
    if match == 5
      % we have an LTPDA database
      ltpda_dbs = [ltpda_dbs cellstr(db)];
    end
  end

end
close(curs);

% If I opened the connection, close it again
if wasEmpty
  close(conn);
end

% Set the list
id = findobj('Tag', 'dbSelect');
set(id, 'String', ltpda_dbs, 'Value', 1);

% Get selected DB
dbh = findobj('Tag', 'dbSelect');
dbs = get(dbh, 'String');
db  = dbs{1};

% set selected DB
dbh = findobj('Tag', 'databaseTxt');
set(dbh, 'String', db);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_importBtn(varargin)

% 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 $
%

% Get connection
mainfig  = findobj('Tag', 'LTPDARepomainfig');
conn     = getappdata(mainfig, 'connection');
if isempty(conn) || ~isa(conn, 'database')
  errordlg('Please connect to a database first', 'No connection found');
  return
end

% Get variable name specs
h = findobj('Tag', 'objPrefixTxt');
prefix = get(h, 'String');
if isempty(prefix)
  prefix = 'obj';
  warning('! Using default prefix (obj)');
end
h = findobj('Tag', 'appendObjTypeChk');
appendObj = get(h, 'Value');

% Get IDs from text box
[ids, cids] = getIds();


%-----------------------------------------------------------------
% Retrieve these ids
objs = [];
for j=1:length(ids)
```

```
  disp(sprintf('+ retrieving object %d', ids(j)));

  % determine object type
  tt = utils.mysql.getObjType(conn, ids(j));
  objname = sprintf('%s%03d', prefix, ids(j));
  if ~isempty(tt) && ~strcmp(tt, 'No Data')
    if appendObj
      objname = [objname '_' tt];
    end
  else
    error('!!! Object type is unknown. Does this object really exist?');
  end

  % Retrieve object
  a = regexp(conn.URL, '//(\S+)/', 'tokens');
  db = regexp(conn.URL, '/', 'split');
  db = db{end};
  % add history
  pl = plist('hostname', a{1}, 'database', db, 'ID', ids(j), 'conn', conn);
  obj = eval(sprintf('%s(pl);', tt));

  assignin('base', objname, obj);
  disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));

  if j==1
    objs = {obj};
  else
    objs = [objs {obj}];
  end
end

disp(sprintf('** Retrieved %d objects.', length(ids)));

%-----------------------------------------------------------------
% Retrieve these Collections
for k=1:length(cids)

  % get Ids from Cid
  ids = utils.mysql.getObjIds(conn, cids(k));
  if isempty(ids)
    error('### This collection doesn''t seem to exist.');
  end

  for j=1:length(ids)
    disp(sprintf('+ retrieving collection %d : %d', cids(k), ids(j)));
    tt = utils.mysql.getObjType(conn, ids(j));
    objname = sprintf('%sC%03d_%03d', prefix, cids(k), ids(j));
    if appendObj
      objname = [objname '_' tt];
    end
    % Retrieve object
    ipbits = regexp(conn.URL, '([0-9]+)', 'match');
    ip = [ipbits{1} '.' ipbits{2} '.' ipbits{3} '.' ipbits{4}];
    db = regexp(conn.URL, '/', 'split');
    db = db{end};
    % add history
    pl = plist('hostname', ip, 'database', db, 'ID', ids(j), 'conn', conn);
    obj = eval(sprintf('%s(pl);', tt));

    assignin('base', objname, obj);
    disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));
    if j==1
      objs = {obj};
    else
      objs = [objs {obj}];
    end
  end
end

disp(sprintf('** Retrieved %d objects.', length(ids)));


end


function [ids, cids] = getIds()

ids  = [];
cids = [];

th = findobj('Tag', 'retrieveIDsTxt');
idStr = get(th, 'String');
cs = cellstr(idStr);

for j=1:length(cs)
  disp('---------')
```

```
  ls = cs{j};
  [s,r] = strtok(ls);
  if ~isempty(s)
    if s(1) == 'c'
      s = s(2:end);
      cids = [cids round(str2num(s))];
    else
      ids = [ids round(str2num(s))];
    end
  end
  while ~isempty(r)
    [s,r] = strtok(r);
    if ~isempty(s)
      if s(1) == 'c'
        s = s(2:end);
        cids = [cids round(str2num(s))];
      else
        ids = [ids round(str2num(s))];
      end
    end
  end
end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_mainfigClose(varargin)

% Callback executed when the main figure closes
%
% M Hewitson
%
% $Id: cb_mainfigClose.m,v 1.3 2008/02/29 10:53:14 hewitson Exp $
%

% Close open connection
conn = getappdata(varargin{1}, 'connection');
if isa(conn, 'database')
  disp('* closing database connection');
  close(conn);
end

disp('* Goodbye from the LTPDA Repository GUI *')
delete(varargin{1})
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_refreshObjsList(varargin)

% Callback to refresh the list of objects.
%
% M Hewitson
%
% $Id: cb_refreshObjsList.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
%


% get a list of objects
objs = getWorkspaceObjs();

% set list
setWorkspaceObjsList(objs);


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_refreshObjsListQL(varargin)

% Callback to refresh the list of objects.
%
% M Hewitson
%
% $Id: cb_refreshObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
%

% get a list of objects
objs = getWorkspaceObjs();

% set list
setWorkspaceObjsListQL(objs);


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_retrieveBtn(varargin)

% 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 $
%

% Get connection
mainfig  = findobj('Tag', 'LTPDARepomainfig');
conn     = getappdata(mainfig, 'connection');
if isempty(conn) || ~isa(conn, 'database')
  errordlg('Please connect to a database first', 'No connection found');
  return
end

% Get variable name specs
h = findobj('Tag', 'objPrefixTxt');
prefix = get(h, 'String');
if isempty(prefix)
  prefix = 'obj';
  warning('! Using default prefix (obj)');
end
h = findobj('Tag', 'appendObjTypeChk');
appendObj = get(h, 'Value');

% Get IDs from text box
[ids, cids] = getIds();


%-----------------------------------------------------------------
% Retrieve these ids
objs = [];
for j=1:length(ids)
```

```
  disp(sprintf('+ retrieving object %d', ids(j)));

  % determine object type
  tt = mysql_getObjType(conn, ids(j));
  objname = sprintf('%s%03d', prefix, ids(j));
  if ~isempty(tt)
    if appendObj
      objname = [objname '_' tt];
    end
  else
    warning('!!! Object type is unknown');
  end

  % Retrieve object
  a = regexp(conn.URL, '//(\S+)/', 'tokens');
  db = regexp(conn.URL, '/', 'split');
  db = db{end};
  % add history
  pl = plist('hostname', a{1}, 'database', db, 'ID', ids(j), 'conn', conn);
  obj = eval(sprintf('%s(pl);', tt));

  assignin('base', objname, obj);
  disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));

  if j==1
    objs = {obj};
  else
    objs = [objs {obj}];
  end
end

disp(sprintf('** Retrieved %d objects.', length(ids)));

%-----------------------------------------------------------------
% Retrieve these Collections
for k=1:length(cids)

  % get Ids from Cid
  ids = mysql_getObjIds(conn, cids(k));

  for j=1:length(ids)
    disp(sprintf('+ retrieving collection %d : %d', cids(k), ids(j)));
    tt = mysql_getObjType(conn, ids(j));
    objname = sprintf('%sC%03d_%03d', prefix, cids(k), ids(j));
    if appendObj
      objname = [objname '_' tt];
    end
    % Retrieve object
    ipbits = regexp(conn.URL, '([0-9]+)', 'match');
    ip = [ipbits{1} '.' ipbits{2} '.' ipbits{3} '.' ipbits{4}];
    db = regexp(conn.URL, '/', 'split');
    db = db{end};
    % add history
    pl = plist('hostname', ip, 'database', db, 'ID', ids(j), 'conn', conn);
    obj = eval(sprintf('%s(pl);', tt));

    assignin('base', objname, obj);
    disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));
    if j==1
      objs = {obj};
    else
      objs = [objs {obj}];
    end
  end
end

disp(sprintf('** Retrieved %d objects.', length(ids)));


end


function [ids, cids] = getIds()

ids  = [];
cids = [];

th = findobj('Tag', 'retrieveIDsTxt');
idStr = get(th, 'String');
cs = cellstr(idStr);

for j=1:length(cs)
  disp('---------')
  ls = cs{j};
  [s,r] = strtok(ls);
  if ~isempty(s)
```

```
    if s(1) == 'c'
      s = s(2:end);
      cids = [cids round(str2num(s))];
    else
      ids = [ids round(str2num(s))];
    end
  end
  while ~isempty(r)
    [s,r] = strtok(r);
    if ~isempty(s)
      if s(1) == 'c'
        s = s(2:end);
        cids = [cids round(str2num(s))];
      else
        ids = [ids round(str2num(s))];
      end
    end
  end
end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_saveBtn(varargin)

% 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 $
%

% Get connection
mainfig  = findobj('Tag', 'LTPDARepomainfig');
conn     = getappdata(mainfig, 'connection');
if isempty(conn) || ~isa(conn, 'database')
  errordlg('Please connect to a database first', 'No connection found');
  return
end

% Get variable name specs
h = findobj('Tag', 'objPrefixTxt');
prefix = get(h, 'String');
if isempty(prefix)
  prefix = 'obj';
  warning('! Using default prefix (obj)');
end
h = findobj('Tag', 'appendObjTypeChk');
appendObj = get(h, 'Value');

% Get IDs from text box
[ids, cids] = getIds();


%-----------------------------------------------------------------
% Retrieve these ids
objs = [];
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.');
0082   end
0083
0084   for j=1:length(ids)
0085     disp(sprintf('+ retrieving collection %d : %d', cids(k), ids(j)));
0086     tt = mysql_getObjType(conn, ids(j));
0087     objname = sprintf('%sC%03d_%03d', prefix, cids(k), ids(j));
0088     if appendObj
0089       objname = [objname '_' tt];
0090     end
0091     % Retrieve object
0092     ipbits = regexp(conn.URL, '([0-9]+)', 'match');
0093     ip = [ipbits{1} '.' ipbits{2} '.' ipbits{3} '.' ipbits{4}];
0094     db = regexp(conn.URL, '/', 'split');
0095     db = db{end};
0096     % add history
0097     pl = plist('hostname', ip, 'database', db, 'ID', ids(j), 'conn', conn);
0098     obj = eval(sprintf('%s(pl);', tt));
0099
0100     assignin('base', objname, obj);
0101     disp(sprintf('** Retrieve object %d to workspace [%s]', ids(j), objname));
0102     if j==1
0103       objs = {obj};
0104     else
0105       objs = [objs {obj}];
0106     end
0107   end
0108 end
0109
0110 disp(sprintf('** Retrieved %d objects.', length(ids)));
0111
0112
0113 end
0114
0115
0116 function [ids, cids] = getIds()
0117
0118 ids  = [];
0119 cids = [];
0120
0121 th = findobj('Tag', 'retrieveIDsTxt');
```

```
idStr = get(th, 'String');
cs = cellstr(idStr);

for j=1:length(cs)
  disp('---------')
  ls = cs{j};
  [s,r] = strtok(ls);
  if ~isempty(s)
    if s(1) == 'c'
      s = s(2:end);
      cids = [cids round(str2num(s))];
    else
      ids = [ids round(str2num(s))];
    end
  end
  while ~isempty(r)
    [s,r] = strtok(r);
    if ~isempty(s)
      if s(1) == 'c'
        s = s(2:end);
        cids = [cids round(str2num(s))];
      else
        ids = [ids round(str2num(s))];
      end
    end
  end
end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_serverSelect(varargin)

% 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 $
%

myh = varargin{1};

mainfig = findobj('Tag', 'LTPDARepomainfig');

% Get server string
servers = get(myh, 'String');
sid     = get(myh, 'Value');

server = servers{sid};

% Set server text field

stxth = findobj('Tag', 'serverTxt');
set(stxth, 'String', server);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_submitBtn(varargin)

% 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 $
%

myh     = varargin{1};
mainfig = varargin{4};

% get object list
objlist = findobj('Tag', 'workspaceObjsList');
objstr  = get(objlist, 'String');
vals    = get(objlist, 'Value');

% Build sinfo structure
sinfo.conn                    = getappdata(mainfig, 'connection');
sinfo.experiment_title        = get(findobj('Tag', 'experimentTitle'), 'String');
sinfo.experiment_description  = get(findobj('Tag', 'experimentDescription'), 'String');
sinfo.analysis_description    = get(findobj('Tag', 'analysisDescription'), 'String');
sinfo.quantity                = get(findobj('Tag', 'quantity'), 'String');
sinfo.keywords                = get(findobj('Tag', 'keywords'), 'String');
sinfo.reference_ids           = get(findobj('Tag', 'referenceIDs'), 'String');
sinfo.additional_comments     = get(findobj('Tag', 'additionalComments'), 'String');
sinfo.additional_authors      = get(findobj('Tag', 'additionalAuthors'), 'String');

if ~isa(sinfo.conn, 'database')
  error('### Please connect to a database before trying to submit.');
end


% Build structure array
k = 1;
objs = {};
for j=1:length(vals)

  objname = deblank(strtok(objstr{vals(j)}));
  cmd = sprintf('obj = evalin(''base'', ''%s'');', objname);
  eval(cmd);

```

```
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)',
objname, mat2str(ids), cid));
0056   end
0057 catch
0058   warning('!!! Failed to submit object: %s', objname);
0059   rethrow(lasterror)
0060 end
0061
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_tableSelect(varargin)

% 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 $
%

myh        = varargin{1};
mainfig    = findobj('Tag', 'LTPDARepomainfig');
fieldsList = findobj('Tag', 'fieldsList');
Nc         = getappdata(mainfig, 'Nconditions');

% Get conn
conn = getappdata(mainfig, 'connection');

% Set fields in list
fields = getFields(conn);
set(fieldsList, 'Value', 1);
set(fieldsList, 'String', fields);

% Set fields in condition drop downs
for j=1:Nc
  cdbh = findobj('Tag', sprintf('c%02dField', j));
  set(cdbh, 'Value', 1);
  set(cdbh, 'String', fields);
end

% Build query
buildquery();

setappdata(mainfig, 'tableSelected', get(myh, 'Value'));

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_workspaceObjsList(varargin)

% 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 $
%


myh     = varargin{1};
mainfig = varargin{4};

% get the selected line
val = get(myh, 'Value');
if length(val) == 1
  lines = get(myh, 'String');
  objname = deblank(strtok(lines{val}));

  cmd = sprintf('obj = evalin(''base'', ''%s'');', objname);
  eval(cmd);

  obj
  objtxt = display(obj);
else
  objtxt = '';
end

st = '';
for j=1:length(objtxt)
  st = [st sprintf('%s\n', objtxt{j})];
end

% set objDisplay Txt
set(myh, 'TooltipString', st);
% ltpdaObjInfo = findobj('Tag', 'ltpdaObjInfo');
% set(ltpdaObjInfo, 'String', objtxt)


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function cb_workspaceObjsListQL(varargin)

% 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 $
%


myh     = varargin{1};
mainfig = varargin{4};

% get the selected line
val = get(myh, 'Value');
if length(val) == 1
  lines = get(myh, 'String');
  objname = deblank(strtok(lines{val}));

  cmd = sprintf('obj = evalin(''base'', ''%s'');', objname);
  eval(cmd);

  objtxt = display(obj);
else
  objtxt = '';
end

% set objDisplay Txt
ltpdaObjInfo = findobj('Tag', 'ltpdaObjInfoQL');
set(ltpdaObjInfo, 'String', objtxt)

fillClassMethodsList();

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function fillClassMethodsList(h)

% Callback to fill the list of class methods
%
% M Hewitson
%
% $Id: fillClassMethodsList.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
%


% Get the object types
id = findobj('Tag', 'workspaceObjsListQL');
ostr = get(id, 'String');
oidx = get(id, 'Value');

objStrs = ostr(oidx);

if ~isempty(objStrs)

  type = getClassFromString(objStrs{1});
  diff = 0;
  for j=2:length(objStrs)
    if ~strcmp(type, getClassFromString(objStrs{j}))
      diff = 1;
    end
  end

  mths = {};
  if ~diff
    tmps = methods(type);
    % Remove exceptions
    for j=1:length(tmps)
```

```
      if ~strcmp(tmps{j}, 'Contents')
%         &&...
%          ~strcmp(tmps{j}, 'subsref')
       mths = [mths tmps(j)];
      end
    end
  end

  % Fill list
  id = findobj('Tag', 'classMethodList');
  set(id, 'Value', 1);
  set(id, 'String', mths);

end

function type = getClassFromString(str)

[s,r] = strtok(str, '(');
type = r(2:end-1);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function fields = getFields(conn)

% Get a list of database fields
%
% M Hewitson
%
% $Id: getFields.m,v 1.3 2008/02/22 09:25:18 hewitson Exp $
%


% get the selected table
tblSel = findobj('Tag', 'tableSelect');
tbls   = get(tblSel, 'String');
tbl    = tbls{get(tblSel, 'Value')};

if ~isempty(conn)
  % get the databases
  try
    q = sprintf('describe %s', tbl);
    curs = exec(conn, q);
    curs = fetch(curs);
    fields = curs.Data(:,1);
    close(curs);
  catch
    disp(sprintf('### Server returned: %s', curs.Message));
    warning('!!! Can not read from database');
    fields = {'-'};
  end
else
  fields = {'-'};
end

```

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 ^

```
function tables = getTables(conn)

% Get a list of database tables
%
% M Hewitson
%
% $Id: getTables.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
%


mainfig = findobj('Tag', 'LTPDARepomainfig');

% Get conn
conn = getappdata(mainfig, 'connection');

if ~isempty(conn)
  % get the databases
  q = 'show tables';
  curs = exec(conn, q);
  curs = fetch(curs);

  tables = curs.Data;
  close(curs);
else
  tables = {'-'};
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function objs = getWorkspaceObjs()

% 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 $
%


% get base workspace variables
ws_vars = evalin('base','whos');

objs = [];
for j=1:length(ws_vars)

  cmd = sprintf('obj = evalin(''base'', ''%s'');', ws_vars(j).name);
  eval(cmd)
  if isa(obj, 'ltpda_uo')
    objs = [objs ws_vars(j)];
  end

end
```

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 ^

```
function setWorkspaceObjsList(objs)

% Fill the workspace object list
%
% M Hewitson
%
% $Id: setWorkspaceObjsList.m,v 1.2 2008/01/22 20:45:22 hewitson Exp $
%


id = findobj('Tag', 'workspaceObjsList');

objlist = [];
for j=1:length(objs)
  obj = objs(j);
  str = sprintf('%s\t\t(%s)', obj.name, obj.class);
  objlist = [objlist cellstr(str)];
end

set(id, 'Value', 1);
set(id, 'String', objlist);


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function setWorkspaceObjsListQL(objs)

% Fill the workspace object list
%
% M Hewitson
%
% $Id: setWorkspaceObjsListQL.m,v 1.1 2008/01/22 21:10:23 hewitson Exp $
%

id = findobj('Tag', 'workspaceObjsListQL');

objlist = [];
for j=1:length(objs)
  obj = objs(j);
  str = sprintf('%s\t\t(%s)', obj.name, obj.class);
  objlist = [objlist cellstr(str)];
end

set(id, 'Value', 1);
set(id, 'String', objlist);


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_lt
- % 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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_launcher_close(varargin)

   % 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 $
   %


   disp('* Goodbye from the LTPDA Launch Bay *')
   delete(varargin{1})



 end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_launcher_launch_ltpdagui(varargin)

    % 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 $
    %

    ltpdagui;


  end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_launcher_launch_pzmodel(varargin)

    % 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 $
    %

    pzmodel_helper;


  end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_launcher_launch_repogui(varargin)

    % 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 $
    %

    repogui;


  end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = ltpdalauncher(varargin)

% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


```

```
%% Check if I exist already
hs = findall(0);
found = -1;
for j=1:length(hs)
  h = hs(j);
  if strcmp(get(h, 'Tag'), 'LTPDAlauncher')
    found = h;
  end
end
if found ~= -1
  figure(found);
  return
end

% id = findobj('Tag', 'LTPDAlauncher');
% if ~isempty(id)
%   figure(id)
%   return
% end

%% Some initial setup

Gproperties.Gcol    = [240 240 240]/255;
Gproperties.Gwidth  = 400;
Gproperties.Gheight = 400;
Gproperties.Gborder = 10;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');

Gproperties.Screen   = get(0,'screensize');
Gproperties.Gposition = [150 ...
  150 ...
  Gproperties.Gwidth...
  Gproperties.Gheight];

%  Initialize and hide the GUI as it is being constructed.
mainfig = figure('Name', 'LTPDA Launch Bay',...
  'NumberTitle', 'off',...
  'Visible','off',...
  'Position',Gproperties.Gposition,...
  'Color', Gproperties.Gcol,...
  'Toolbar', 'none',...
  'Resize', 'off',...
  'ToolBar',      'none',            ...
  'NextPlot',      'new',            ...
  'HandleVisibility', 'callback', ...
  'MenuBar', 'none',...
  'Tag', 'LTPDAlauncher');

% Set mainfig callbacks
set(mainfig, 'CloseRequestFcn', {'ltpda_launcher_close', mainfig});

% Set Application data
setappdata(mainfig, 'Gproperties', Gproperties);

%% Add  Logo
M = 0.01;
ax   = axes('Position', [M M 1-2*M 1-2*M], 'Parent', mainfig);
logo = imread('images/ltpda_blue.jpg');
image(logo, 'Parent', ax)
set(ax,'Visible', 'on', 'Box', 'off', 'Xtick', [], 'Ytick', []);
axis(ax, 'off');

%% Add launch buttons

BtnHeight = 20;
BtnWidth  = Gproperties.Gwidth - 20;
BtnLeft   = 10;
BtnBottom = Gproperties.Gheight - BtnHeight - 10;

%-- LTPDA GUI
BtnBottom = BtnBottom - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','LTPDA GUI',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpda_launcher_ltpdagui');
set(pbh, 'Callback', {'ltpda_launcher_launch_ltpdagui'});

%-- LTPDA REPO GUI
pos       = get(pbh, 'Position');
BtnBottom = pos(2) - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','LTPDA REPO GUI',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpda_launcher_repogui');
set(pbh, 'Callback', {'ltpda_launcher_launch_repogui'});

%-- PZmodel helper
pos       = get(pbh, 'Position');
```

```
BtnBottom = pos(2) - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','pzmodel helper',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpda_launcher_pzmodel');
set(pbh, 'Callback', {'ltpda_launcher_launch_pzmodel'});

%-- specwin helper
pos       = get(pbh, 'Position');
BtnBottom = pos(2) - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','specwin GUI',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpda_launcher_specwin');
set(pbh, 'Callback', {'specwin_viewer'});

%-- constructor helper
pos       = get(pbh, 'Position');
BtnBottom = pos(2) - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','Constructor Helper',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpda_constructor_helper');
set(pbh, 'Callback', {'ltpda_constructor_helper'});

%-- object explorer
pos       = get(pbh, 'Position');
BtnBottom = pos(2) - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','Object Explorer',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpda_object_explorer');
set(pbh, 'Callback', {@call_explore_ao});

%-- quicklook
pos       = get(pbh, 'Position');
BtnBottom = pos(2) - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','Quicklook',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpdaquicklook');
set(pbh, 'Callback', {'ltpdaquicklook'});

%-- ltpdv
pos       = get(pbh, 'Position');
BtnBottom = pos(2) - BtnHeight - 10;
pbh = uicontrol(mainfig,'Style','pushbutton','String','ltpdv',...
  'Position',[BtnLeft BtnBottom BtnWidth BtnHeight ],...
  'Tag', 'ltpdv');
set(pbh, 'Callback', {'ltpdv'});

%-- version string
v  = ver('LTPDA');

sth = uicontrol(mainfig,'Style','text',...
  'String',sprintf('%s %s', v.Name, v.Version),...
  'HorizontalAlignment', 'left', ...
  'BackgroundColor', [130 190 250]/255, ...
  'ForegroundColor', 'w', ...
  'Position',[10 10 120 25]);




%% Start the GUI

% Make the GUI visible.
set(mainfig,'Visible','on')


  function call_explore_ao(varargin)

    ltpda_explorer;

  end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv(varargin)
  % 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
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


  %% Check if I exist already
  id = findobj('Tag', 'LTPDVmainfig');
  if ~isempty(id)
    figure(id)
    return
  end

  %% Some initial setup

  Gproperties.Gcol     = [240 240 240]/255;
  Gproperties.Gwidth   = 1100;
  Gproperties.Gheight  = 700;
  Gproperties.Gborder  = 10;
  Gproperties.fontsize = 12;

  Gproperties.Screen   = get(0,'screensize');
```

```
Gproperties.Gposition = [Gproperties.Screen(3)/2-Gproperties.Gwidth/2 ...
  Gproperties.Screen(4)/2-Gproperties.Gheight/2 ...
  Gproperties.Gwidth...
  Gproperties.Gheight];

%  Initialize and hide the GUI as it is being constructed.
mainfig = figure('Name', 'LTPDA Dataviewer',...
  'NumberTitle', 'off',...
  'Visible','off',...
  'Position',Gproperties.Gposition,...
  'Color', Gproperties.Gcol,...
  'Toolbar', 'none',...
  'MenuBar', 'none',...
  'Resize', 'on',...
  'Tag', 'LTPDVmainfig');

% Set mainfig callbacks
set(mainfig, 'CloseRequestFcn', {'ltpdv_cb_mainfigClose', mainfig});

% Set Application data
setappdata(mainfig, 'Gproperties', Gproperties);

%% Setup GUI elements

Hmargin = Gproperties.Gborder/Gproperties.Gwidth;
Vmargin = Gproperties.Gborder/Gproperties.Gheight;
BottomMargin = 0.2;
SplitPoint   = 0.35;

%----------- Object list

% Panel
pleft = Hmargin;
phght = 1-2*Vmargin;
pbot  = (1 - Vmargin - phght);
pwid  = SplitPoint - 2*Hmargin;
ph =  uipanel(mainfig, 'Title','Object List','FontSize', Gproperties.fontsize,...
  'BackgroundColor', [0.6 0.7 0.9],...
  'Position',[pleft pbot pwid phght], 'Units', 'normalized');

% Object list
pmarg   = 0.025;
Lleft   = pmarg;
Lbottom = 0.4+pmarg;
Lwidth  = 1-2*pmarg;
Lheight = 1-Lbottom-pmarg;
th = uitable(ph, ...
  'Units', 'normalized',...
  'ColumnName', {'Name','Type'}, ...
  'ColumnWidth', {220, 100}, ...
  'ColumnEditable', [true false], ...
  'Position', [Lleft Lbottom Lwidth Lheight],...
  'Tag', 'LTPDVobjList', ...
  'CellSelectionCallback', {@ltpdv_objectList_select, mainfig}, ...
  'CellEditCallback', {@ltpdv_objectList_edit, mainfig});

% Set column widths
set(th, 'Units', 'pixel');
pos = get(th, 'Position');
set(th, 'Units', 'normalized');
set(th, 'ColumnWidth', {0.7*pos(3), 0.2*pos(3)});

% Delete objects button
pleft = pmarg;
phght = 0.05;
pbott = Lbottom - phght - pmarg;
pwidt = 0.2;
pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Delete',...
  'Callback', {@ltpdv_delete_objects, mainfig}, ...
  'Units', 'normalized', ...
  'Position',[pleft pbott pwidt phght]);

% Clear objects button
pleft = pmarg + pwidt + pmarg;
phght = 0.05;
pbott = Lbottom - phght - pmarg;
pwidt = 0.2;
pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Clear',...
  'Callback', {@ltpdv_delete_objects, mainfig, 'all'}, ...
  'Units', 'normalized', ...
  'Position',[pleft pbott pwidt phght]);

% Save objects button
pleft = pleft + pwidt + pmarg;
```

```
phght = 0.05;
pbott = Lbottom - phght - pmarg;
pwidt = 0.2;
pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Save',...
  'Callback', {@ltpdv_save_objects, mainfig}, ...
  'Units', 'normalized', ...
  'Position',[pleft pbott pwidt phght]);

% Load objects button
pleft = pleft + pwidt + pmarg;
phght = 0.05;
pbott = Lbottom - phght - pmarg;
pwidt = 0.2;
pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Load',...
  'Callback', {@ltpdv_load_objects, mainfig}, ...
  'Units', 'normalized', ...
  'Position',[pleft pbott pwidt phght]);

% Export objects button
pleft = pmarg;
phght = 0.05;
pbott = pbott - phght - pmarg;
pwidt = 0.2;
pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Export',...
  'Callback', {@ltpdv_export_objects, mainfig}, ...
  'Units', 'normalized', ...
  'Position',[pleft pbott pwidt phght]);

% Explore objects button
pleft = pleft + pwidt + pmarg;
pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Explore',...
  'Callback', {@ltpdv_explore_objects, mainfig}, ...
  'Units', 'normalized', ...
  'Position',[pleft pbott pwidt phght]);


%-------
% Plot button
bw = 0.2;
bl = pmarg;
bh = 0.05;
bb = pmarg;

pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Plot',...
  'Callback', {@ltpdv_plot, mainfig}, ...
  'Units', 'normalized', ...
  'Position',[bl bb bw bh]);

% Plot history button
bw = 0.2;
bl = bl+bw+pmarg;
bh = 0.05;
bb = pmarg;

pbh = uicontrol(ph,'Style','pushbutton',...
  'String','Plot History',...
  'Callback', {@ltpdv_hist_plot, mainfig}, ...
  'Units', 'normalized', ...
  'Position',[bl bb bw bh]);


%----------- TAB PANEL
PanLeft = SplitPoint;
Tleft   = PanLeft + Hmargin;
Theight = 1-2*Vmargin ;
Tbottom = (1 - Vmargin - Theight);
Twidth  = (1-PanLeft)-2*Hmargin;

htab = uitabgroup('Parent', mainfig, ...
  'Units','normalized',...
  'Position',[Tleft Tbottom Twidth Theight], 'SelectionChangeFcn',
{@ltpdv_tabChanged});
t1 = uitab(htab, 'title', 'Import');
t2 = uitab(htab, 'title', 'Math');
t3 = uitab(htab, 'title', 'Preprocessing');
t4 = uitab(htab, 'title', 'Filter');
t5 = uitab(htab, 'title', 'Spectral');
t6 = uitab(htab, 'title', 'Object Properties');
t7 = uitab(htab, 'title', 'Plot');

% 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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

make_ltpda_objects

PURPOSE ^

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
clear all;

%% FSDATA
p1 = pz(1);
p2 = pz(4, 20);
z1 = pz(20);
pzm1 = pzmodel(2.3, [p1 p2], z1);
r = resp(pzm1, plist('f1', 0.01, 'f2', 100, 'nf', 1000, 'scale', 'log'));

%% TSDATA
fs = 10;
nsecs = 1000;

ats = ao(plist('tsfcn', 'randn(size(t))', 'fs', fs, 'nsecs', nsecs));

%% CDATA

ac = ao(1:100);

%% SPECWIN
s = specwin('Hanning', 1000);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = scratch(varargin)
% 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

% Edit the above text to modify the response to help scratch

% Last Modified by GUIDE v2.5 26-Aug-2008 16:10:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @scratch_OpeningFcn, ...
                   'gui_OutputFcn',  @scratch_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before scratch is made visible.
function scratch_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to scratch (see VARARGIN)

% Choose default command line output for scratch
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes scratch wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = scratch_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function hh = uitabpanel(varargin)

% 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 $
%
%


% Author: Shiying Zhao (zhao@arch.umsl.edu)
% Version: 1.0
% First created: May 02, 2006
% Last modified: May 20, 2006

tag = '';
units = 'normalized';
position = [0,0,1,1];
fontweight = 'normal';
[style,type] = deal('lefttop',2);
visible = 'on';

foregroundcolor = [1,1,1];
backgroundcolor = [0.7725,0.8431,0.9608];
panelbackgroundcolor = [0.4314,0.5882,0.8431];
titlehighlightcolor = [1,0.8,0];

horizontalalignment = 'center';
[createfcn,deletefcn,resizefcn] = deal({},{},{});
```

```

active = 0;
styles = {'popup','lefttop','righttop','leftbottom','rightbottom', ...
  'centertop','centerbottom'};

for k=1:2:nargin
  property = lower(varargin{k});
  value = varargin{k+1};
  switch property
    case {'style','tabposition','titleposition'}
      style = value;
      type = strmatch(lower(style),styles);
      if isempty(type)
        error(['Bad value for Style/TitlePosition property: ' style '.']);
      end
    case 'fontsize'
      fontsize = value;
    case {'title','string'}
      string = value;
      ntab = length(string);
    case {'frameposition','position'}
      position = value;
    case {'framebackgroundcolor','backgroundcolor'}
      backgroundcolor = value;
    case {'framebordertype','bordertype'}
      bordertype = value;
    case {'panelheights','heights'}
      panelheight = value;
    case {'titleforegroundcolor','foregroundcolor'}
      foregroundcolor = value;
    case {'selecteditem','active'}
      active = value;
    otherwise
      eval([property '= value;']);
  end
end

if ~exist('parent','var')
  parent = gcf;
end
if ~exist('margins','var')
  if type==1
    margins = {[1.5,0.75,1.5,1],'characters'};
  else
    margins = {1,'pixels'};
  end
end
if ~iscell(margins)
  margins = {margins,units};
end

if ~exist('bordertype','var')
  if type==1
    bordertype = 'etchedin';
  else
    bordertype = 'none';
  end
end

if ~exist('titlebackgroundcolor','var')
  titlebackgroundcolor = 0.95*panelbackgroundcolor;
end

cmenu = uicontextmenu;
uimenu(cmenu,'Label','Goto Tab');
uimenu(cmenu,'Separator','on');

htab = uipanel( ...
  'Parent',parent, ...
  'Units',units, ...
  'Position',position, ...
  'DeleteFcn',deletefcn, ...
  'ResizeFcn',resizefcn, ...
  'BackgroundColor',backgroundcolor, ...
  'BorderType',bordertype, ...
  'UIContextMenu',cmenu, ...
  'Visible',visible, ...
  'FontSize', fontsize,...
  'Tag',tag);
set(htab,'Units','pixels');
tabpos = get(htab,'Position');

status = uicontainer( ...
  'Parent',htab, ...
  'BackgroundColor',backgroundcolor, ...
  'Units',margins{2}, ...
```

```
  'Position',[1,1,1,1], ...
  'UIContextMenu',cmenu, ...
  'Visible','off');
set(status,'Units','pixels');
charsz = get(status,'Position');
margins = [charsz(1:2),charsz(1:2)].*margins{1};
set(status,'Units','characters');
set(status,'Position',[1,1,1,1]);
set(status,'Units','pixels');
charsz = get(status,'Position');
titleheight = 2*charsz(4);

if type==1
  %-------------------------------------------------------------------------
  %  PopupTab
  %-------------------------------------------------------------------------
  panelpos = tabpos;
  titlepos = [margins(1),tabpos(2)+tabpos(4)-titleheight-margins(2), ...
    tabpos(3)-sum(margins([1,3])),titleheight];
  ttextpos = [0.05,0.1,0.9,0.76];

  if ~exist('panelheight','var')
    panelheight = ones(1,ntab);
  end
  if ~exist('panelbordertype','var')
    panelbordertype = 'line';
  end

  for k=1:ntab
    title(k) = uicontrol( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'Position',titlepos, ...
      'ForegroundColor',foregroundcolor, ...
      'BackgroundColor',titlebackgroundcolor, ...
      'FontWeight',fontweight, ...
      'FontSize', fontsize,...
      'HorizontalAlignment',horizontalalignment, ...
      'Callback',{@PopupTabCbk,k}, ...
      'Style','toggle', ...
      'String',string{k}, ...
      'Value',0, ...
      'Visible','on');

    panel(k) = uipanel( ...
      'Parent',htab, ...
      'Units','pixels', ...
      'Position',titlepos+[0,0,0,panelheight(k)*charsz(4)], ...
      'BackgroundColor',panelbackgroundcolor, ...
      'BorderType',panelbordertype, ...
      'Visible','off');

    uimenu(cmenu, ...
      'Label',string{k}, ...
      'Callback',{@PopupTabCbk,k});
  end

  uimenu(cmenu,'Separator','on');
  uimenu(cmenu, ...
    'Label','Expand All', ...
    'Callback',{@PopupTabEnableAllCbk,'on'});
  uimenu(cmenu, ...
    'Label','Collapse All', ...
    'Callback',{@PopupTabEnableAllCbk,'off'});

  sliderpos = [tabpos(3)-14,2,15,tabpos(4)-3];
  slider = uicontrol( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'Position',sliderpos, ...
    'BackgroundColor',panelbackgroundcolor, ...
    'Style','slider', ...
    'Callback',@PopupTabSliderCbk, ...
    'Value',0, ...
    'Min',-1, ...
    'Max',0, ...
    'Visible','off');

  statuspos = titlepos;
  statuspos([2,4]) = [margins(4),panelpos(4)-(ntab+1)*(margins(4)+titlepos(4))];
  set(status,'Position',statuspos);
  PopupTabCbk(title(1),[],active);
  set(status,'ResizeFcn',@PopupTabResizeCbk);

else
  %-------------------------------------------------------------------------
```

```
%  TopT/Bottom Tab
%-----------------------------------------------------------------------------
margins(3:4) = -(margins(1:2)+margins(3:4));
switch type
  case 2 %lefttop
    [loop,sign] = deal(1:ntab,[1,0,1,0]);
  case 3 %righttop
    [loop,sign] = deal(ntab:-1:1,[ntab,0,0,0]);
  case 4 %lefttbottom
    [loop,sign] = deal(1:ntab,[1,0,1,1]);
  case 5 %rightbottom
    [loop,sign] = deal(ntab:-1:1,[ntab,0,0,1]);
  case 6 %centertop
    [loop,sign] = deal(1:ntab,[1,ntab,1,0]);
  case 7 %centerbottom
    [loop,sign] = deal(1:ntab,[1,ntab,1,1]);
end
addtitle = 2*[sign(3:4),-1,-1];
if sign(3)
  ttextpos = [0.05,0.08,0.9,0.76];
else
  ttextpos = [0.05,0.05,0.9,0.76];
end

if ~exist('panelbordertype','var')
  panelbordertype = 'beveledout';
end

for k=1:ntab
  title(k) = uipanel( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'BackgroundColor',titlebackgroundcolor, ...
    'BorderType',panelbordertype, ...
    'ButtonDownFcn',{@TopBottomTabCbk,k}, ...
    'Visible','on');
  %titlewidth(k) = (length(string{k})+4)*1.2*charsz(3);

  ttext(k) = uicontrol( ...
    'Parent',title(k), ...
    'Units','normalized', ...
    'Position',ttextpos, ...
    'ForegroundColor',foregroundcolor, ...
    'BackgroundColor',titlebackgroundcolor, ...
    'HorizontalAlignment',horizontalalignment, ...
    'Style','text', ...
    'String',string{k}, ...
    'Enable','inactive', ...
    'FontSize', fontsize,...
    'HitTest','off',...
    'Visible','on');

  panel(k) = uipanel( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'BackgroundColor',panelbackgroundcolor, ...
    'BorderType',panelbordertype, ...
    'Visible','off');

  uimenu(cmenu, ...
    'Label',string{k}, ...
    'Callback',{@TopBottomTabCbk,k});
end

if ~active, active = 1; end
set(panel(active),'Visible','on');

inset(1) = uicontrol( ...
  'Parent',htab, ...
  'Units','pixels', ...
  'BackgroundColor',panelbackgroundcolor, ...
  'Style','text', ...
  'Visible','on');
if ~isnan(titlehighlightcolor)
  inset(2) = uicontrol( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'BackgroundColor',titlehighlightcolor, ...
    'Style','text', ...
    'Visible','on');
  inset(3) = uicontrol( ...
    'Parent',htab, ...
    'Units','pixels', ...
    'BackgroundColor',0.85*titlehighlightcolor, ...
    'Style','text', ...
    'Visible','on');
```

```
  end

  TopBottomTabResizeCbk(status,[]);
  set(status,'ResizeFcn',@TopBottomTabResizeCbk);
end

setappdata(htab,'panels',panel);
setappdata(htab,'status',status);
if ~isempty(createfcn)
  createfcn(htab,[],panel,status);
end

if nargout, hh = htab; end


%-------------------------------------------------------------------------
% Inner Functions for Popup Tab
%-------------------------------------------------------------------------
  function PopupTabCbk(hobj,evdt,indx)

    set(title,'Value',0);
    if indx>0
      indxvis = strcmp(get(panel(indx),'Visible'),'off');
      if indxvis
        set(panel(indx),'Visible','on');
      else
        set(panel(indx),'Visible','off');
      end
    end
    if isempty(strmatch('on',get(panel,'Visible')))
      set(status,'Visible','on');
    else
      set(status,'Visible','off');
    end

    currval = get(slider,'Value');
    if currval<0
      set(slider,'Value',0);
      PopupTabSliderCbk(slider,evdt);
    end
    titlepos = get(title(1),'Position');
    addtolen = titlepos(2)+titlepos(4)+margins(4);

    for k=1:ntab
      titlepos = get(title(k),'Position');
      titlepos(2) = addtolen-titlepos(4)-margins(4);
      set(title(k),'Position',titlepos);
      addtolen = titlepos(2);
      visible = strcmp(get(panel(k),'Visible'),'on');
      if visible
        panelpos = get(panel(k),'Position');
        addtolen = addtolen-panelpos(4);
        panelpos(2) = addtolen;
        set(panel(k),'Position',panelpos);
      end
    end

    if visible
      botpos = get(panel(ntab),'Position');
    else
      botpos = get(title(ntab),'Position');
    end

    visible = strcmp(get(slider,'Visible'),'on');
    addwidth = (-1)^(~visible);
    if botpos(2)*addwidth>0
      if visible
        set(slider,'Visible','off');
      else
        set(slider,'Visible','on');
      end
    else
      if visible
        addwidth = 0;
      else
        return;
      end
    end

    addwidth = addwidth*sliderpos(3);
    for k=1:ntab
      titlepos = get(title(k),'Position');
      titlepos(3) = titlepos(3)+addwidth;
      set(title(k),'Position',titlepos);
      set(title(k),'UserData',titlepos(2));

```

```
      panelpos = get(panel(k),'Position');
      panelpos(3) = panelpos(3)+addwidth;
      set(panel(k),'Position',panelpos);
      set(panel(k),'UserData',panelpos(2));
    end

    toplen = botpos(2)-margins(2);
    if toplen>0
      set(slider,'Value',0);
      return;
    end

    try
      if indxvis
        viewpos = get(panel(indx),'Position');
        if viewpos(2)-viewpos(4)<0
          if viewpos(4)+margins(4)<=tabpos(4);
            currval = min(0,viewpos(2)-margins(4)/2);
          else
            viewpos = get(title(indx),'Position');
            currval = viewpos(2)+viewpos(4)+margins(4)/2-tabpos(4);
          end
        end
      end
    catch
    end

    set(slider,'Min',toplen);
    set(slider,'Value',max(currval,toplen));
    set(slider,'SliderStep',[0.3,1]/max(1,-toplen/100));
    PopupTabSliderCbk(slider,evdt);

  end


%-------------------------------------------------------------------------
  function PopupTabSliderCbk(hobj,evdt)

    currval = get(hobj,'Value');
    for k=1:ntab
      titlepos = get(title(k),'Position');
      titlepos(2) = get(title(k),'UserData')-currval;
      set(title(k),'Position',titlepos);
      panelpos = get(panel(k),'Position');
      panelpos(2) = get(panel(k),'UserData')-currval;
      set(panel(k),'Position',panelpos);
    end

  end


%-------------------------------------------------------------------------
  function PopupTabEnableAllCbk(hobj,evdt,cstr)

    set(panel,'Visible',cstr);
    PopupTabCbk(hobj,evdt,0);

  end


%-------------------------------------------------------------------------
  function PopupTabResizeCbk(hobj,evdt)

    oldpos = tabpos;
    units = get(htab,'Units');
    set(htab,'Units','pixels');
    tabpos = get(htab,'Position');
    set(htab,'Units',units);

    addtopos = tabpos-oldpos;
    sliderpos([1,2,4]) = sliderpos([1,2,4])+addtopos([1,2,4]);
    set(slider,'Position',sliderpos);

    addtopos = [tabpos(3:4)-oldpos(3:4),0,0];
    currval = get(slider,'Value');
    if currval<0
      set(slider,'Value',0);
      PopupTabSliderCbk(slider,evdt);
    end

    for k=1:ntab
      titlepos = get(title(k),'Position');
      set(title(k),'Position',titlepos+addtopos);
    end
    PopupTabCbk(title(1),evdt,0);

```

```
    visible = strcmp(get(slider,'Visible'),'on');
    if visible & currval<0
      set(slider,'Value',max(currval,get(slider,'Min')));
      PopupTabSliderCbk(slider,evdt);
    end

  end


%-----------------------------------------------------------------------------
% Inner Functions for Top/Bottom Tab
%-----------------------------------------------------------------------------
  function TopBottomTabCbk(hobj,evdt,indx)

    if active==indx, return; end

    % deactivate tab(active)
    n = active;
    titlepos = get(title(n),'Position');
    titlepos = titlepos+addtitle.*[n==sign(1),1,n==sign(1),1];
    titlepos(3) = titlepos(3)+addtitle(3)*(n==sign(2));
    set(title(n),'Position',titlepos,'BackgroundColor',titlebackgroundcolor);
    set(ttext(n),'FontWeight','normal','BackgroundColor',titlebackgroundcolor);
    set(panel(n),'Visible','off');

    % activate tab(indx)
    active = indx;
    titlepos = get(title(indx),'Position');
    titlepos = titlepos-addtitle.*[indx==sign(1),1,indx==sign(1),1];
    titlepos(3) = titlepos(3)-addtitle(3)*(indx==sign(2));

    set(title(indx),'Position',titlepos,'BackgroundColor',panelbackgroundcolor);
    set(ttext(indx),'FontWeight','bold','BackgroundColor',panelbackgroundcolor);
    insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
    set(inset(1),'Position',insetpos);
    try
      insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
      set(inset(2),'Position',insetpos);
      set(inset(3),'Position',insetpos+[2,1-2*sign(4),-4,0]);
    catch
    end
    set(panel(indx),'Visible','on');

    if isappdata(htab,'SelectionChangeFcn')
      SelectionChangeFcn = getappdata(htab,'SelectionChangeFcn');
      eventdata = struct('EventName','SelectionChanged', ...
        'OldValue',n,'NewValue',indx,'Panels',panel,'Status',status);
      if iscell(SelectionChangeFcn)
        SelectionChangeFcn{1}(hobj,eventdata,SelectionChangeFcn{2:end});
      else
        SelectionChangeFcn(hobj,eventdata);
      end
    end

  end


%-----------------------------------------------------------------------------
  function TopBottomTabResizeCbk(hobj,evdt)

    units = get(htab,'Units');
    set(htab,'Units','pixels');
    tabpos = get(htab,'Position');
    set(htab,'Units',units);

    titlepos = [0,0,0,titleheight];
    if sign(4)
      panelpos = [0,titleheight+1,tabpos(3),tabpos(4)-titleheight]+margins;
      titlepos(2) = panelpos(2)-titlepos(4);
    else
      panelpos = [0,0,tabpos(3),tabpos(4)-titleheight]+margins;
      titlepos(2) = panelpos(2)+panelpos(4);
    end
    set(panel,'Position',panelpos);

    n = sum(cellfun('length',string))+2*ntab;
    if type<6
      if (n+2*ntab)*1.2*charsz(3)+2>panelpos(3)
        [loop,sign(1:3)] = deal(1:ntab,[1,ntab,1]);
      else
        if type==2 | type==4
          [loop,sign(1:3)] = deal(1:ntab,[1,0,1]);
        else
          [loop,sign(1:3)] = deal(ntab:-1:1,[ntab,0,0]);
        end
      end
```

```
      addtitle = 2*[sign(3:4),-1,-1];
    end
    titlepos(1) = panelpos(1)+(1-sign(3))*panelpos(3);
    for k=loop
      if sign(2)
        titlepos(3) = panelpos(3)*(length(string{k})+2)/n;
        addtopos = addtitle.*[k==1,1,k==1|k==ntab,1];
      else
        titlepos(3) = (length(string{k})+4)*1.2*charsz(3);
        addtopos = addtitle.*[k==sign(1),1,k==sign(1),1];
      end
      titlepos(1) = titlepos(1)-(1-sign(3))*titlepos(3);
      set(title(k),'Position',titlepos+addtopos);
      titlepos(1) = titlepos(1)+sign(3)*titlepos(3);
    end

    statuspos = titlepos;
    statuspos(1) = sign(3)*titlepos(1)+2;
    statuspos(3) = titlepos(1)+sign(3)*(tabpos(3)-2*titlepos(1))-4;

    titlepos = get(title(active),'Position');
    titlepos = titlepos-addtitle.*[active==sign(1),1,active==sign(1),1];
    titlepos(3) = titlepos(3)-addtitle(3)*(active==sign(2));
    set(title(active),'Position',titlepos,'BackgroundColor',panelbackgroundcolor);
    set(ttext(active),'FontWeight','bold','BackgroundColor',panelbackgroundcolor);

    insetpos = [titlepos(1)+1,titlepos(2)+sign(4)*titlepos(4)-2,titlepos(3)-1,3];
    set(inset(1),'Position',insetpos);
    try
      insetpos = insetpos+[0,(1-2*sign(4))*titleheight+2*sign(4),0,-2];
      set(inset(2),'Position',insetpos);
      set(inset(3),'Position',insetpos+[2,1-2*sign(4),-4,0]);
    catch
    end
    try
      % This causes the above statements being excuted twice for each resizing.
      set(status,'Position',statuspos,'Visible','on');
    catch
      set(status,'Visible','off');
    end

  end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_buildPreprocessPan
- 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_selec
-  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_selec
-  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
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filterBrowse">m/gui/ltpdv/callbacks/ltpdv_filterBrowse</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filterDelete">m/gui/ltpdv/callbacks/ltpdv_filterDelete</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filterImport">m/gui/ltpdv/callbacks/ltpdv_filterImport</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filter_control_chans">m/gui/ltpdv/callbacks/ltpdv_filter_control_chan
- DV_FILTERCONTROLCHANS filter out control channels
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filter_from_standard_panel">m/gui/ltpdv/callbacks/ltpdv_filter_from_s
-  builds a filter from the description on
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filter_get_pzmodel">m/gui/ltpdv/callbacks/ltpdv_filter_get_pzmodel</a
-  poles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filter_getf">m/gui/ltpdv/callbacks/ltpdv_filter_getf</a>
-  f1
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filter_list_select">m/gui/ltpdv/callbacks/ltpdv_filter_list_select</a
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_filtersClear">m/gui/ltpdv/callbacks/ltpdv_filtersClear</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_getWorkspaceObjs">m/gui/ltpdv/callbacks/ltpdv_getWorkspaceObjs</a>
-  gets a list of LTPDA user objects currently in the
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_get_params_from_panel">m/gui/ltpdv/callbacks/ltpdv_get_params_from_pa
-  get a plist from the parameters on a
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_get_selected_filter">m/gui/ltpdv/callbacks/ltpdv_get_selected_filter
-  selected panel
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_get_selected_objs">m/gui/ltpdv/callbacks/ltpdv_get_selected_objs</a>
-  gets the selected objects from the object list.
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_hist_plot">m/gui/ltpdv/callbacks/ltpdv_hist_plot</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_importObjs">m/gui/ltpdv/callbacks/ltpdv_importObjs</a>
-  import given objects to the object list
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_load_objects">m/gui/ltpdv/callbacks/ltpdv_load_objects</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_objectList_edit">m/gui/ltpdv/callbacks/ltpdv_objectList_edit</a>
-  activates when the user changes the content of a
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_objectList_select">m/gui/ltpdv/callbacks/ltpdv_objectList_select</a>
-  activates when the user selects cells in the
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_parse_control_chans">m/gui/ltpdv/callbacks/ltpdv_parse_control_chans
- DV_PARSECONTROLCHANS parse out a list of control channels from the input
   <a href="matlab:help m/gui/ltpdv/callbacks/ltpdv_plot">m/gui/ltpdv/callbacks/ltpdv_plot</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_plotFilterResp">m/gui/ltpdv/callbacks/ltpdv_plotFilterResp</a>
-  plots the response of the selected filter.
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_poleAdd">m/gui/ltpdv/callbacks/ltpdv_poleAdd</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_poleClear">m/gui/ltpdv/callbacks/ltpdv_poleClear</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_poleDelete">m/gui/ltpdv/callbacks/ltpdv_poleDelete</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_preproc_fcnlist">m/gui/ltpdv/callbacks/ltpdv_preproc_fcnlist</a>
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_preproc_param_act">m/gui/ltpdv/callbacks/ltpdv_preproc_param_act</a>
- Handles
   m/gui/ltpdv/callbacks/ltpdv_preproc_sets_callback       - (No help available)
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_properties_apply">m/gui/ltpdv/callbacks/ltpdv_properties_apply</a>
-  callback for apply button on the object properties
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_properties_build_props">m/gui/ltpdv/callbacks/ltpdv_properties_build_
- % Object class
   <a href="matlab:help
```

```
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<
-  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_s
- 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<
-  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_late
-  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_
- Handle
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_get_stop_time">m/gui/ltpdv/callbacks/ltpdv_server_get_stop_ti
- 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_sele
- Handles
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_set_channel_list">m/gui/ltpdv/callbacks/ltpdv_server_set_chan
- Get normal channels
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_set_duration">m/gui/ltpdv/callbacks/ltpdv_server_set_duration<
- Get handle
   <a href="matlab:help
m/gui/ltpdv/callbacks/ltpdv_server_set_timeslist">m/gui/ltpdv/callbacks/ltpdv_server_set_timeslis
- 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_fcnlist">m/gui/ltpdv/callbacks/ltpdv_spectral_fcnlist</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_sele
-  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_se
-  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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_buildFilterPanel(panH, mainfig)

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
```

```
bl = bl + bw + pmarg;
bb = 2*pmarg+bh;
sth = uicontrol(panH, 'Style','text',...
   'String', 'Num f',...
   'Units', 'normalized', ...
   'BackgroundColor', 'w', ...
   'Fontsize', 12, ...
   'Position',[bl bb tw bh]);
bl = bl+tw+pmarg;
sth = uicontrol(panH, 'Style','edit',...
   'String', '',...
   'Units', 'normalized', ...
   'BackgroundColor', 'w', ...
   'Fontsize', 12, ...
   'Tag', 'LTPDVfilterNf', ...
   'Position',[bl bb bw bh]);

% scale
bb = pmarg;
bl = bl -tw-pmarg;
sth = uicontrol(panH, 'Style','text',...
   'String', 'Scale',...
   'Units', 'normalized', ...
   'BackgroundColor', 'w', ...
   'Fontsize', 12, ...
   'Position',[bl bb tw bh]);
bl = bl+tw+pmarg;
sth = uicontrol(panH, 'Style','popupmenu',...
   'String', {'log', 'lin'},...
   'Units', 'normalized', ...
   'BackgroundColor', 'w', ...
   'Fontsize', 12, ...
   'Tag', 'LTPDVfilterScale', ...
   'Position',[bl bb bw bh]);


%-----------------------------------------------------------------------------
% Build filter pole/zero panel
   function buildFilterPoleZeroPanel(panH, mainfig)

      pmarg = 0.025;

      % freq entry
      bl = pmarg;
      bw = 0.15;
      tw = 0.2;
      bh = 0.05;
      bb = 1 - bh - pmarg;
      sth = uicontrol(panH, 'Style','text',...
         'String', 'Freq [Hz]',...
         'Units', 'normalized', ...
         'BackgroundColor', 'w', ...
         'Fontsize', 12, ...
         'Position',[bl bb tw bh]);
      bl = bl+tw+pmarg;
      sth = uicontrol(panH, 'Style','edit',...
         'String', '',...
         'Units', 'normalized', ...
         'BackgroundColor', 'w', ...
         'Fontsize', 12, ...
         'Tag', 'LTPDVfilterPZfreq', ...
         'Position',[bl bb bw bh]);

      % q entry
      bl = bl + bw + pmarg;
      bb = 1 - bh - pmarg;
      sth = uicontrol(panH, 'Style','text',...
         'String', 'Q',...
         'Units', 'normalized', ...
         'BackgroundColor', 'w', ...
         'Fontsize', 12, ...
         'Position',[bl bb tw/2 bh]);
      bl = bl+tw+pmarg;
      sth = uicontrol(panH, 'Style','edit',...
         'String', '',...
         'Units', 'normalized', ...
         'BackgroundColor', 'w', ...
         'Fontsize', 12, ...
         'Tag', 'LTPDVfilterPZq', ...
         'Position',[bl bb bw bh]);

      %----------- Poles
      pl = pmarg;
      pw = 0.5 - 2*pmarg;
      ph = 0.6;
      pb = bb - pmarg - ph;
```

```
        poleh = uipanel(panH, 'Title', 'Poles', ...
           'Units', 'normalized', ...
           'Position', [pl pb pw ph]);
        % add btn
        bl = pmarg;
        bh = 0.07;
        bb = 1-bh-pmarg;
        bw = 1-2*pmarg;
        pbh = uicontrol(poleh,'Style','pushbutton',...
           'String','Add',...
           'Callback', {@ltpdv_poleAdd, htab, mainfig}, ...
           'Units', 'normalized', ...
           'Position',[bl bb bw bh]);

        % list
        ll = pmarg;
        lh = 0.7;
        lb = bb-lh-pmarg;
        lw = 1-2*pmarg;
        lh = uicontrol(poleh, 'Style', 'listbox', ...
           'String', ' ',...
           'BackgroundColor', 'w', ...
           'Value', 1, 'Units', 'normalized', ...
           'Fontsize', 14, ...
           'Tag', 'LTPDVfilterPolelist', ...
           'max', 1000, ...
           'Position',[ll lb lw lh]);

        % del btn
        bl = pmarg;
        bb = lb-pmarg-bh;
        pbh = uicontrol(poleh,'Style','pushbutton',...
           'String','Delete',...
           'Callback', {@ltpdv_poleDelete, htab, mainfig}, ...
           'Units', 'normalized', ...
           'Position',[bl bb bw bh]);
        % clear btn
        bl = pmarg;
        bb = bb-bh;
        pbh = uicontrol(poleh,'Style','pushbutton',...
           'String','Clear',...
           'Callback', {@ltpdv_poleClear, htab, mainfig}, ...
           'Units', 'normalized', ...
           'Position',[bl bb bw bh]);

        %----------- Zeros
        pl = 0.5 + pmarg;
        zeroh = uipanel(panH, 'Title', 'Zeros', ...
           'Units', 'normalized', ...
           'Position', [pl pb pw ph]);
        % Add
        bl = pmarg;
        bh = 0.07;
        bb = 1-bh-pmarg;
        bw = 1-2*pmarg;
        pbh = uicontrol(zeroh,'Style','pushbutton',...
           'String','Add',...
           'Callback', {@ltpdv_zeroAdd, htab, mainfig}, ...
           'Units', 'normalized', ...
           'Position',[bl bb bw bh]);
        % list
        ll = pmarg;
        lh = 0.7;
        lb = bb-lh-pmarg;
        lw = 1-2*pmarg;
        lh = uicontrol(zeroh, 'Style', 'listbox', ...
           'String', ' ',...
           'BackgroundColor', 'w', ...
           'Value', 1, 'Units', 'normalized', ...
           'Fontsize', 14, ...
           'Tag', 'LTPDVfilterZerolist', ...
           'max', 1000, ...
           'Position',[ll lb lw lh]);
        % del btn
        bl = pmarg;
        bb = lb-pmarg-bh;
        pbh = uicontrol(zeroh,'Style','pushbutton',...
           'String','Delete',...
           'Callback', {@ltpdv_zeroDelete, htab, mainfig}, ...
           'Units', 'normalized', ...
           'Position',[bl bb bw bh]);
        % clear btn
        bl = pmarg;
        bb = bb-bh;
        pbh = uicontrol(zeroh,'Style','pushbutton',...
           'String','Clear',...
```

```
            'Callback', {@ltpdv_zeroClear, htab, mainfig}, ...
            'Units', 'normalized', ...
            'Position',[bl bb bw bh]);

        % Gain entry
        pos = get(zeroh, 'Position');
        bl = pmarg;
        bw = 0.15;
        tw = 0.15;
        bh = 0.06;
        bb = pos(2) - bh - pmarg;
        sth = uicontrol(panH, 'Style','text',...
            'String', 'Gain',...
            'Units', 'normalized', ...
            'BackgroundColor', 'w', ...
            'Fontsize', 12, ...
            'Position',[bl bb tw bh]);
        bl = bl+tw+pmarg;
        sth = uicontrol(panH, 'Style','edit',...
            'String', '',...
            'Units', 'normalized', ...
            'BackgroundColor', 'w', ...
            'Fontsize', 12, ...
            'Tag', 'LTPDVfilterPZgain', ...
            'Position',[bl bb bw bh]);

        % Name entry
        bl = bl + bw + pmarg;
        sth = uicontrol(panH, 'Style','text',...
            'String', 'Name',...
            'Units', 'normalized', ...
            'BackgroundColor', 'w', ...
            'Fontsize', 12, ...
            'Position',[bl bb tw bh]);
        bl = bl+tw+pmarg;
        bw = 0.3;
        sth = uicontrol(panH, 'Style','edit',...
            'String', '',...
            'Units', 'normalized', ...
            'BackgroundColor', 'w', ...
            'Fontsize', 12, ...
            'Tag', 'LTPDVfilterPZname', ...
            'Position',[bl bb bw bh]);

        % Pole/Zero resp button
        bb = bb - bh - pmarg;
        bl = pmarg;
        bw = 0.22;
        pbh = uicontrol(panH,'Style','pushbutton',...
            'String','Plot response',...
            'Callback', {@ltpdv_pzmResp, panH, mainfig}, ...
            'Units', 'normalized', ...
            'Position',[bl bb bw bh]);


        % Pole/Zero store button
        bl = bl+bw+pmarg;
        pbh = uicontrol(panH,'Style','pushbutton',...
            'String','Store model',...
            'Callback', {@ltpdv_pzmStore, panH, mainfig}, ...
            'Units', 'normalized', ...
            'Position',[bl bb bw bh]);

        % Pole/Zero store resp button
        bl = bl+bw+pmarg;
        pbh = uicontrol(panH,'Style','pushbutton',...
            'String','Store model resp',...
            'Callback', {@ltpdv_pzmStoreResp, panH, mainfig}, ...
            'Units', 'normalized', ...
            'Position',[bl bb bw bh]);

    end
%-------------------------------------------------------------------------
% Build filter object panel
    function buildFilterObjectPanel(panH, mainfig)

        pmarg = 0.025;

        % import button
        bl = pmarg;
        bh = 0.04;
        bw = 0.2;
        bb = 1 - bh - pmarg;
        pbh = uicontrol(panH,'Style','pushbutton',...
            'String','Import Filter',...
            '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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_buildImportPanel(ph, mainfig)

  Gproperties = getappdata(mainfig, 'Gproperties');

  pmarg = 0.025;
  % Build a panel to contain this tab
  % panh = uipanel('Parent', ph, 'Position', [pmarg pmarg 1-2*pmarg 1-2*pmarg]);

  %% Add new sub tabbed panel

  Hmargin = 0.025;
  Vmargin = 0.025;

  PanLeft = Hmargin;
  Tleft   = Hmargin;
  Theight = 1-2*Vmargin;
  Tbottom = (1 - Theight - Vmargin);
  Twidth  = 1-2*Hmargin;

  Pcol = [1 1 0.9];

  htab = uitabgroup('Parent', ph, ...
    'Units','normalized',...
    'Tag', 'LTPDVimportTabPanel', ...
    'Position',[Tleft Tbottom Twidth Theight], 'SelectionChangeFcn',
{@ltpdv_tabChanged});
  mt1 = uitab(htab, 'title', 'Server');
  mt2 = uitab(htab, 'title', 'Workspace');
  mt3 = uitab(htab, 'title', 'Build');

  %-------------- Build Server Panel
  buildServerPanel(mt1, mainfig);
  %-------------- Build Workspace Panel
  h = buildWorkspacePanel(mt2, mainfig);
  %-------------- Build Build Panel
  buildBuildPanel(mt3, mainfig);
end

%-------------------------------------------------------------------------
% Build import from server panel
function buildBuildPanel(hp, mainfig)


  EditFontSize = 11;
  vmarg   = 0.03;
  pmarg   = 0.01;
  CommHeight = EditFontSize/400;
  ph = 1;
  htab = uipanel('Parent', hp, 'Position', [pmarg 1-ph-pmarg 1-2*pmarg ph]);
  set(htab, 'Units', 'Normalized')
  set(htab, 'Title', '')
  set(htab, 'BackgroundColor', [0.6 0.6 0.8]);
  set(htab, 'ForegroundColor', 'w');


  %----- Class selector
  bh = CommHeight;
  bl = pmarg;
  bw = 0.15;
  bb = 1-bh-1*vmarg;
  sth = uicontrol(htab, 'Style','text',...
    'String', 'Object Class',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw bh]);
  bl   = bl+bw+pmarg;
  bw  = 0.2;
  clsh = uicontrol(htab, ...
    'Style', 'popupmenu', ...
    'Units', 'normalized',...
    'Fontsize', EditFontSize, ...
    'BackgroundColor', 'w', ...
    'String', utils.helper.ltpda_userclasses, ...
    'Position', [bl bb bw bh],...
    'Tag', 'LTPDVbuildObjectClass', ...
    'Callback', {@ltpdv_build_class_select, mainfig});

  %----- Set selector
  bl = bl+bw+pmarg;
  bw = 0.15;
  sth = uicontrol(htab, 'Style','text',...
```

```
    'String', 'Parameter Set',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw bh]);
  bl   = bl+bw+pmarg;
  bw  = 0.3;
  dtypeh = uicontrol(htab, ...
    'Style', 'popupmenu', ...
    'Units', 'normalized',...
    'Fontsize', EditFontSize, ...
    'BackgroundColor', 'w', ...
    'String', ' ', ...
    'Position', [bl bb bw bh],...
    'Tag', 'LTPDVbuildSetSelect', ...
    'Callback', {@ltpdv_build_set_select, mainfig});

  % Parameter panel
  ph = 0.8;
  ppan = uipanel('Parent', htab, 'Position', [pmarg 0.08+vmarg 1-2*pmarg ph]);
  set(ppan, 'Units', 'Normalized')
  set(ppan, 'Title', 'Parameters')
  set(ppan, 'BackgroundColor', [0.7 0.7 0.8]);
  set(ppan, 'ForegroundColor', 'w');
  set(ppan, 'Tag', 'LTPDVbuildParamsPanel');

  pbh = uicontrol(htab,'Style','pushbutton',...
    'String','Build object',...
    'Callback', {@ltpdv_build_buildobject, mainfig}, ...
    'Units', 'normalized', ...
    'Fontsize', EditFontSize, ...
    'Position',[pmarg vmarg 0.15 2*bh]);

  % call back
  ltpdv_build_class_select(clsh, mainfig)

%
%   bh = CommHeight;
%   bl = pmarg;
%   bw = 1-2*pmarg;
%   bb = vmarg;
%   sth = uicontrol(hp, 'Style','text',...
%     'String', 'Build objects with the ''constructor helper'' then import from workspace.',...
%     'Units', 'normalized', ...
%     'BackgroundColor', 'w', ...
%     'Fontsize', EditFontSize, ...
%     'Position',[bl bb bw bh]);

end
%-----------------------------------------------------------------------------
% Build import from server panel
function buildServerPanel(hp, mainfig)

  % Data type
  EditFontSize = 11;
  vmarg   = 0.03;
  pmarg   = 0.01;
  CommHeight = EditFontSize/80;

  % Import button
  bw = 0.12;
  bh = 0.05;
  bl = pmarg;
  bb = pmarg;
  pbh = uicontrol(hp,'Style','pushbutton',...
    'String','Import',...
    'Callback', {@ltpdv_server_import, mainfig}, ...
    'Units', 'normalized', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw bh]);

  %======== New tab panel
  Tleft = pmarg;
  Theight =  0.8;
  Tbottom = 1-Theight - pmarg;
  Twidth = 1-2*pmarg;
  %     htab = uitabgroup('Parent', hp, ...
  %       'Units','normalized',...
  %       'Position',[Tleft Tbottom Twidth Theight], 'SelectionChangeFcn', {@ltpdv_tabChanged});
  %     t1 = uitab(htab, 'title', 'Server Details');
  %     t2 = uitab(htab, 'title', 'Times');
  %     t3 = uitab(htab, 'title', 'Channels');

  % ======= Server Details
```

```

ph = 0.3;
htab = uipanel('Parent', hp, 'Position', [pmarg 1-ph-pmarg 1-2*pmarg ph]);
set(htab, 'Units', 'Normalized')
set(htab, 'Title', 'Server settings')
set(htab, 'BackgroundColor', [0.3 0.7 0.3]);
set(htab, 'ForegroundColor', 'w');

bh = CommHeight;
bl = pmarg;
bw = 0.1;
bb = 1-bh-3*vmarg;
sth = uicontrol(htab, 'Style','text',...
  'String', 'Data type',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
bl   = bl+bw+pmarg;
bw  = 0.17;
dtypeh = uicontrol(htab, ...
  'Style', 'popupmenu', ...
  'Units', 'normalized',...
  'Fontsize', EditFontSize, ...
  'BackgroundColor', 'w', ...
  'String', {'frame data','h(t) frames', 'raw data', 'hour trends', 'day trends'},
...
  'Position', [bl bb bw bh],...
  'Tag', 'LTPDVserverDataType', ...
  'Callback', {@ltpdv_server_datatype_select, mainfig});

% Statistic
bl = bl + bw + pmarg;
bw = 0.1;
sth = uicontrol(htab, 'Style','text',...
  'String', 'Statistic',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
bl   = bl+bw+pmarg;
bw  = 0.12;
th = uicontrol(htab, ...
  'Style', 'popupmenu', ...
  'Units', 'normalized',...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'String', {'rms','min', 'max', 'avg'}, ...
  'Position', [bl bb bw bh],...
  'Tag', 'LTPDVserverStat');

% Server drop-down box
bl = pmarg;
bw = 0.1;
bb = bb - bh - vmarg;
sth = uicontrol(htab, 'Style','text',...
  'String', 'Server',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
bl   = bl+bw+pmarg;
bw  = 0.25;
th = uicontrol(htab, ...
  'Style', 'popupmenu', ...
  'Units', 'normalized',...
  'BackgroundColor', 'w', ...
  'String', {'130.75.117.159','130.75.117.73', '130.75.117.164', 'localhost'}, ...
  'Fontsize', EditFontSize, ...
  'Position', [bl bb bw bh],...
  'Tag', 'LTPDVserverServer', ...
  'Callback', {@ltpdv_server_server_select, mainfig});

% Server entry box
bl   = bl+bw+pmarg;
bw  = 0.22;
th = uicontrol(htab, ...
  'Style', 'edit', ...
  'Units', 'normalized',...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'String', '', ...
  'Position', [bl bb bw bh],...
  'Tag', 'LTPDVserverServerEdit');

% Port entry
```

```
bl = bl+bw+pmarg;
bw = 0.07;
sth = uicontrol(htab, 'Style','text',...
  'String', 'Port',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
bl   = bl+bw+pmarg;
bw  = 0.1;
th = uicontrol(htab, ...
  'Style', 'edit', ...
  'Units', 'normalized',...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'String', '', ...
  'Position', [bl bb bw bh],...
  'Tag', 'LTPDVserverPortEdit');

% RDS
bl = pmarg;
bw = 0.1;
bb = bb - bh - vmarg;
sth = uicontrol(htab, 'Style','text',...
  'String', 'RDS Level',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
bl   = bl+bw+pmarg;
bw  = 0.05;
th = uicontrol(htab, ...
  'Style', 'edit', ...
  'Units', 'normalized',...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'String', '', ...
  'Position', [bl bb bw bh],...
  'Tag', 'LTPDVserverRdsEdit');

% CAL
bl = pmarg + bl + bw;
bw = 0.1;
sth = uicontrol(htab, 'Style','text',...
  'String', 'Cal Ver',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
bl   = bl+bw+pmarg;
bw  = 0.05;
th = uicontrol(htab, ...
  'Style', 'edit', ...
  'Units', 'normalized',...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'String', '', ...
  'Position', [bl bb bw bh],...
  'Tag', 'LTPDVserverCalEdit');


% call callbacks
ltpdv_server_server_select(0, mainfig);
ltpdv_server_datatype_select(dtypeh, mainfig);

% Query server button
bw = 0.13;
bb = vmarg;
bl = pmarg;
pbh = uicontrol(htab,'Style','pushbutton',...
  'String','Query Server',...
  'Callback', {@ltpdv_server_queryServer, mainfig}, ...
  'Units', 'normalized', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw 1.4*CommHeight]);

% Get latest button
bl = bl + bw + pmarg;
pbh = uicontrol(htab,'Style','pushbutton',...
  'String','Get latest',...
  'Callback', {@ltpdv_server_getLatest, mainfig}, ...
  'Units', 'normalized', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw 1.4*CommHeight]);


```

```
%---------------- Time entry

ph = 0.3;
htab = uipanel('Parent', hp, 'Position', [pmarg 1-2*ph-2*pmarg 1-2*pmarg ph]);
set(htab, 'Units', 'Normalized')
set(htab, 'Title', 'Times settings')
set(htab, 'BackgroundColor', [0.7 0.3 0.3]);
set(htab, 'ForegroundColor', 'w');

% time and channel lists
bb = 1-0.7*bh-pmarg;
bl = pmarg+0.4;
bw = 0.5;
timesH = uicontrol(htab, 'Style','text',...
  'String', 'Times',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw 0.7*bh]);

bl = pmarg + 0.4;
bh = 0.7;
bb = bb - bh;
lh = uicontrol(htab, 'Style', 'listbox', ...
  'String', '',...
  'BackgroundColor', 'w', ...
  'Value',1, 'Units', 'normalized', ...
  'Fontsize', EditFontSize, ...
  'Max', 1000, ...
  'Tag', 'LTPDVtimesList', ...
  'Position',[bl bb bw bh], ...
  'Callback', {@ltpdv_server_timeslist, mainfig});

% delete times
bw = (0.5 - 3*pmarg)/ 4;
bh = CommHeight;
bb = bb - bh - vmarg;
pbh = uicontrol(htab,'Style','pushbutton',...
  'String','Delete times',...
  'Callback', {@ltpdv_server_deleteTimes, mainfig}, ...
  'Units', 'normalized', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
% clear list
bl = bl + pmarg + bw;
pbh = uicontrol(htab,'Style','pushbutton',...
  'String','Clear list',...
  'Callback', {@ltpdv_server_clearTimes, mainfig}, ...
  'Units', 'normalized', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
% save list
bl = bl + pmarg + bw;
pbh = uicontrol(htab,'Style','pushbutton',...
  'String','Save list',...
  'Callback', {@ltpdv_server_saveTimes, mainfig}, ...
  'Units', 'normalized', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
% load list
bl = bl + pmarg + bw;
pbh = uicontrol(htab,'Style','pushbutton',...
  'String','Load list',...
  'Callback', {@ltpdv_server_loadTimes, mainfig}, ...
  'Units', 'normalized', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);

% start time
pos = get(timesH, 'Position');
bl = pmarg;
bw = 0.1;
bh = CommHeight;
bb = pos(2)-2*vmarg;
sth = uicontrol(htab, 'Style','text',...
  'String', 'Start',...
  'Units', 'normalized', ...
  'BackgroundColor', 'w', ...
  'Fontsize', EditFontSize, ...
  'Position',[bl bb bw bh]);
bl   = bl+bw+pmarg;
bw  = 0.25;
th = uicontrol(htab, ...
  'Style', 'edit', ...
  'Units', 'normalized',...
  'BackgroundColor', 'w', ...
```

```
    'Fontsize', EditFontSize, ...
    'String', '', ...
    'Position', [bl bb bw bh],...
    'Callback', {@ltpdv_server_start_edit, mainfig}, ...
    'Tag', 'LTPDVserverStartEdit');
  % stop time
  bl = pmarg;
  bw = 0.1;
  bb = bb - bh - vmarg;
  sth = uicontrol(htab, 'Style','text',...
    'String', 'Stop',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw bh]);
  bl   = bl+bw+pmarg;
  bw  = 0.25;
  th = uicontrol(htab, ...
    'Style', 'edit', ...
    'Units', 'normalized',...
    'BackgroundColor', 'w', ...
    'String', '', ...
    'Fontsize', EditFontSize, ...
    'Position', [bl bb bw bh],...
    'Callback', {@ltpdv_server_stop_edit, mainfig}, ...
    'Tag', 'LTPDVserverStopEdit');

  % duration display
  bw = 0.1;
  bl = pmarg;
  bb = bb - bh - vmarg;
  sth = uicontrol(htab, 'Style','text',...
    'String', 'Duration',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw bh]);
  bl = bl + bw + pmarg;
  bw = 0.25;
  sth = uicontrol(htab, 'Style','text',...
    'String', '',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Tag', 'LTPDVserverDurationDisplay', ...
    'Position',[bl bb bw bh]);

  % comment
  pos = get(sth, 'Position');
  bl = pmarg;
  bw = pos(1) + pos(3) - pmarg;
  bb = bb - 0.7*bh - vmarg;
  sth = uicontrol(htab, 'Style','text',...
    'String', 'Comment',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw bh*0.7]);
  bb = bb - bh - pmarg;
  bl = pmarg;
  th = uicontrol(htab, ...
    'Style', 'edit', ...
    'Units', 'normalized',...
    'BackgroundColor', 'w', ...
    'String', '', ...
    'Fontsize', EditFontSize, ...
    'Position', [bl bb bw bh],...
    'Tag', 'LTPDVserverCommentEdit');

  % Add time button
  bl = pmarg;
  bb = bb - bh - vmarg;
  pbh = uicontrol(htab,'Style','pushbutton',...
    'String','Add to list',...
    'Fontsize', EditFontSize, ...
    'Callback', {@ltpdv_server_addTime, mainfig}, ...
    'Units', 'normalized', ...
    'Position',[bl bb bw bh]);


  % ----------------- Channel list
  ph = 0.3;
  htab = uipanel('Parent', hp, 'Position', [pmarg pmarg+2*vmarg 1-2*pmarg ph]);
  set(htab, 'Units', 'Normalized')
  set(htab, 'Title', 'Channel settings')
  set(htab, 'BackgroundColor', [0.3 0.3 0.7]);
```

```
  set(htab, 'ForegroundColor', 'w');

  bw = 0.5;
  bb = 1 - vmarg - 0.7*bh;
  bl = pmarg;
  sth = uicontrol(htab, 'Style','text',...
    'String', 'Channels',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw 0.7*bh]);
  bh = 0.8;
  bb = bb - bh;
  lh = uicontrol(htab, 'Style', 'listbox', ...
    'String', '',...
    'BackgroundColor', 'w', ...
    'Value',1, 'Units', 'normalized', ...
    'Fontsize', EditFontSize, ...
    'Max', 1000, ...
    'Tag', 'LTPDVchannelList', ...
    'Position',[bl bb bw bh], ...
    'Callback', {@ltpdv_server_channellist, mainfig});

  % include control chans
  %     bw = 0.3;
  %     bh = CommHeight;
  %     bb = bb - bh - pmarg;
  %     th = uicontrol(htab, ...
  %       'Style', 'checkbox', ...
  %       'Units', 'normalized',...
  %       'BackgroundColor', 'w', ...
  %       'String', 'Include control', ...
  %       'Fontsize', EditFontSize, ...
  %       'Position', [bl bb bw bh],...
  %       'ForegroundColor', 'w',...
  %       'Tag', 'LTPDVchannelIncludeControl');

  % search box
  bl = 0.55;
  bw = 0.08;
  bh = CommHeight;
  bb = 1 - vmarg - bh;
  sth = uicontrol(htab, 'Style','text',...
    'String', 'Search',...
    'Units', 'normalized', ...
    'BackgroundColor', 'w', ...
    'Fontsize', EditFontSize, ...
    'Position',[bl bb bw bh]);
  bl   = bl+bw+pmarg;
  bw  = 0.17;
  th = uicontrol(htab, ...
    'Style', 'edit', ...
    'Units', 'normalized',...
    'BackgroundColor', 'w', ...
    'String', '', ...
    'Fontsize', EditFontSize, ...
    'Position', [bl bb bw bh],...
    'Callback', {@ltpdv_search_channels, mainfig}, ...
    'Tag', 'LTPDVchannelSearchEdit');
  pos = get(sth, 'Position');
  bl = pos(1);
  bb = bb - bh - vmarg;
  pbh = uicontrol(htab,'Style','pushbutton',...
    'String','Get channels',...
    'Fontsize', EditFontSize, ...
    'Callback', {@ltpdv_server_getChannels, mainfig}, ...
    'Units', 'normalized', ...
    'Position',[bl bb bw bh]);
end
%-------------------------------------------------------------------------
% Build import from workspace panel
function th = buildWorkspacePanel(hp, mainfig)

  % Object list
  pmarg   = 0.025;
  Lleft   = pmarg;
  Lbottom = 0.4+pmarg;
  Lwidth  = 1-2*pmarg;
  Lheight = 1-Lbottom-pmarg;
  th = uitable(hp, ...
    'Units', 'normalized',...
    'ColumnName', {'Name','Type', 'Size', 'kB'}, ...
    'Position', [Lleft Lbottom Lwidth Lheight],...
    'Tag', 'LTPDVworkspaceList', ...
    'CellSelectionCallback', {@ltpdv_workspaceList_select, mainfig});

```

```
  % set data
  ltpdv_workspaceList_set_data(th);

  % Make refresh button
  pl = pmarg;
  ph = 0.05;
  pb = Lbottom - ph - pmarg;
  pw = 0.1;
  pbh = uicontrol(hp,'Style','pushbutton',...
    'String','Refresh',...
    'Callback', {@ltpdv_workspaceList_set_data, th}, ...
    'Units', 'normalized', ...
    'Position',[pl pb pw ph]);

  % Make import button
  pl = pmarg + pw + pmarg;
  ph = 0.05;
  pb = Lbottom - ph - pmarg;
  pw = 0.1;
  pbh = uicontrol(hp,'Style','pushbutton',...
    'String','Import',...
    'Callback', {@ltpdv_importObjs, mainfig, th}, ...
    'Units', 'normalized', ...
    'Position',[pl pb pw ph]);

  % Selection box for renamin with var names
  pl = pl + pw + pmarg;
  ph = 0.05;
  pb = Lbottom - ph - pmarg;
  pw = 0.3;
  ch = uicontrol(hp, 'Style', 'checkbox', ...
    'String', 'use var name', ...
    'TooltipString', 'Set object name to workspace variable name.', ...
    'Units', 'normalized', ...
    'Position', [pl pb pw ph], ...
    'Value', 1, ...
    'Tag', 'LTPDVusevarnameChk');

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_buildMathPanel(panH, mainfig)

Gproperties = getappdata(mainfig, 'Gproperties');

% Sizes
pmarg   = 0.025;            % Margin size
pw      = 0.5 - 2*pmarg;    % Panel width
ph      = 0.3;             % Panel height
bheight = 0.2;              % Button height
bwidth  = 0.2;             % Button width
BUTTON_FONT = 12;
%--------------------- Calculator pad
pl = pmarg;
pb = 0.5 + pmarg;
% Panel
pan =  uipanel(panH, 'Title','Calculator  ','FontSize', Gproperties.fontsize,...
   'BackgroundColor', [0.4 0.4 0.4],...
   'ForegroundColor', 'w',...
   'Position',[pl pb pw ph], 'Units', 'normalized');

% Buttons
buts = {'+', '-', './', '.*', '.^', '*', '/', '^'};
bl = pmarg;
bb = 1-pmarg-bheight;
for j=1:numel(buts)
   bh = uicontrol(pan, 'String',buts{j},...
      'Callback', {@ltpdv_calc_btn, mainfig, 2}, ...
      'Units', 'normalized', 'Fontsize', BUTTON_FONT, ...
      'Position',[bl bb bwidth bheight]);
   % move
   bl = bl + bwidth + pmarg;
   if bl + bwidth > (1-pmarg)
      bl = pmarg;
      bb = bb - bheight - pmarg;
   end
end

%------------------------ Stats panel
pl = 0.5 + pmarg;
pb = 0.5 + pmarg;
% Panel
pan =  uipanel(panH, 'Title','Stats  ','FontSize', Gproperties.fontsize,...
   'BackgroundColor', [0.4 0.4 0.4],...
   'ForegroundColor', 'w',...
   'Position',[pl pb pw ph], 'Units', 'normalized');

% Buttons
```

```
buts = {'mean', 'median', 'sum', 'std', 'var'};
bl = pmarg;
bb = 1-pmarg-bheight;
for j=1:numel(buts)
   bh = uicontrol(pan, 'String',buts{j},...
      'Callback', {@ltpdv_calc_btn, mainfig, 1}, ...
      'Units', 'normalized', 'Fontsize', BUTTON_FONT, ...
      'Position',[bl bb bwidth bheight]);
   % move
   bl = bl + bwidth + pmarg;
   if bl + bwidth > (1-pmarg)
      bl = pmarg;
      bb = bb - bheight - pmarg;
   end
end

%------------------------ Trig panel
pl = pmarg;
pb = pmarg;
% Panel
pan =  uipanel(panH, 'Title','Trig  ','FontSize', Gproperties.fontsize,...
   'BackgroundColor', [0.4 0.4 0.4],...
   'ForegroundColor', 'w',...
   'Position',[pl pb pw ph], 'Units', 'normalized');

% Buttons
buts = {'sin', 'asin', 'cos', 'acos', 'tan', 'atan', 'log', 'log10', 'exp'};
bl = pmarg;
bb = 1-pmarg-bheight;
for j=1:numel(buts)
   bh = uicontrol(pan, 'String',buts{j},...
      'Callback', {@ltpdv_calc_btn, mainfig, 1}, ...
      'Units', 'normalized', 'Fontsize', BUTTON_FONT, ...
      'Position',[bl bb bwidth bheight]);
   % move
   bl = bl + bwidth + pmarg;
   if bl + bwidth > (1-pmarg)
      bl = pmarg;
      bb = bb - bheight - pmarg;
   end
end

%------------------------ Misc panel
pl = 0.5 + pmarg;
pb = pmarg;
% Panel
pan =  uipanel(panH, 'Title','Misc  ','FontSize', Gproperties.fontsize,...
   'BackgroundColor', [0.4 0.4 0.4],...
   'ForegroundColor', 'w',...
   'Position',[pl pb pw ph], 'Units', 'normalized');

% Buttons
buts = {'sqrt', 'real', 'imag', 'conj', 'abs', 'phase'};
bl = pmarg;
bb = 1-pmarg-bheight;
for j=1:numel(buts)
   bh = uicontrol(pan, 'String',buts{j},...
      'Callback', {@ltpdv_calc_btn, mainfig, 1}, ...
      'Units', 'normalized', 'Fontsize', BUTTON_FONT, ...
      'Position',[bl bb bwidth bheight]);
   % move
   bl = bl + bwidth + pmarg;
   if bl + bwidth > (1-pmarg)
      bl = pmarg;
      bb = bb - bheight - pmarg;
   end
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpdv_buildPlotPanel

PURPOSE ^

SYNOPSIS ^

function ltpdv_buildPlotPanel(panH, mainfig)

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpdv_buildPlotPanel(panH, mainfig)

Gproperties = getappdata(mainfig, 'Gproperties');

pmarg = 0.025;
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_buildPreprocessPanel(panH, mainfig)

Gproperties = getappdata(mainfig, 'Gproperties');

% List of functions
preFcns = {'delay';...
   'downsample'; ...
   'dsmean'; ...
   'firwhiten'; ...
   'interp'; ...
   'join'; ...
   'polyfit'; ...
   'resample'; ...
   'select'; ...
   'smoother'; ...
   'split'; ...
   'timeshift'; ...
   'upsample'};

% Margins
pmarg = 0.025;
FONTSIZE = 12;

% Text string
ll = pmarg;
lh = 0.05;
lb = 1-pmarg-lh;
lw = 0.3;
sth = uicontrol(panH,'Style','text',...
   'String','Functions',...
   'Units', 'normalized', ...
   'BackgroundColor', Gproperties.Gcol, ...
   'Fontsize', 14, ...
   'Position',[ll lb lw lh]);

% Params panel
pl = pmarg + 0.3 + pmarg;
pb = 0.1;
pw = 1-pl-pmarg;
ph = 1 - pb - pmarg;

paramh = uipanel('Parent',panH,'Title','Parameters',...
   'Units', 'normalized', ...
   'BackgroundColor', 0.8*[0.87 0.92 0.99], ...
   'Fontsize', FONTSIZE, ...
```

```
    'Tag', 'LTPDVpreprocparampan', ...
    'Position',[pl pb pw ph]);

% Function list
ll = pmarg;
lb = pmarg;
lw = 0.3;
lh = 1-2*pmarg - 0.05;
lh = uicontrol(panH, 'Style', 'listbox', ...
    'String', preFcns,...
    'BackgroundColor', 'w', ...
    'Value',1, 'Units', 'normalized', ...
    'Fontsize', FONTSIZE, ...
    'Tag', 'LTPDVpreprocfcnlist', ...
    'Position',[ll lb lw lh], ...
    'Callback', {@ltpdv_preproc_fcnlist, paramh, mainfig});

% Build params
ltpdv_preproc_fcnlist(lh, paramh, mainfig);

% Process button
phght = 0.05;
pbott = pmarg;
pwidt = 0.2;
pleft = 1-pmarg-pwidt;
pbh = uicontrol(panH,'Style','pushbutton',...
    'String','Process',...
    'Callback', {@ltpdv_fcn_run, 'preproc', panH, mainfig}, ...
    'Units', 'normalized', ...
    'Position',[pleft pbott pwidt phght]);



% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpdv_buildPropertiesPanel

PURPOSE ^

SYNOPSIS ^

function ltpdv_buildPropertiesPanel(panH, mainfig)

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

SOURCE CODE ^

```
function ltpdv_buildPropertiesPanel(panH, mainfig)

  EditFontSize = 11;
  vmarg   = 0.03;
  pmarg   = 0.01;
  CommHeight = EditFontSize/400;
  ph = 1-2*vmarg;
  htab = uipanel('Parent', panH, 'Position', [pmarg vmarg 1-2*pmarg ph]);
  set(htab, 'Units', 'Normalized')
  set(htab, 'Title', 'Properties for ')
  set(htab, 'BackgroundColor', [1 1 0.6]);
  set(htab, 'ForegroundColor', 'k');
  set(htab, 'Tag', 'LTPDVobjectProperties');



end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_fcnlist Handles

This function is called by:

SOURCE CODE ^

```
function ltpdv_buildSpectralPanel(panH, mainfig)


Gproperties = getappdata(mainfig, 'Gproperties');

% List of functions
preFcns = {'psd';...
           'lpsd'; ...
           'tfe'; ...
           'ltfe'; ...
           'cohere'; ...
           'lcohere'; ...
           'cpsd'; ...
           'lcpsd'; ...
           'linedetect'; ...
           'rms'; ...
                     };

% Margins
pmarg = 0.025;

% Text string
ll = pmarg;
lh = 0.05;
lb = 1-pmarg-lh;
lw = 0.3;
sth = uicontrol(panH,'Style','text',...
   'String','Functions',...
   'Units', 'normalized', ...
   'BackgroundColor', Gproperties.Gcol, ...
   'Fontsize', 14, ...
   'Position',[ll lb lw lh]);

% Params panel
pl = pmarg + 0.3 + pmarg;
pb = 0.1;
pw = 1-pl-pmarg;
ph = 1 - pb - pmarg;

paramh = uipanel('Parent',panH,'Title','Parameters',...
   'Units', 'normalized', ...
   'BackgroundColor', 0.8*[0.99 0.92 0.82], ...
   'Fontsize', 14, ...
   'Tag', 'LTPDVspectralparampan', ...
   'Position',[pl pb pw ph]);
```

```
% Function list
ll = pmarg;
lb = pmarg;
lw = 0.3;
lh = 1-2*pmarg - 0.05;
lh = uicontrol(panH, 'Style', 'listbox', ...
   'String', preFcns,...
   'BackgroundColor', 'w', ...
   'Value',1, 'Units', 'normalized', ...
   'Fontsize', 14, ...
   'Tag', 'LTPDVspectralfcnlist', ...
   'Position',[ll lb lw lh], ...
   'Callback', {@ltpdv_spectral_fcnlist, paramh, mainfig});

% Build params
ltpdv_spectral_fcnlist(lh, paramh, mainfig);

% Process button
phght = 0.05;
pbott = pmarg;
pwidt = 0.2;
pleft = 1-pmarg-pwidt;
pbh = uicontrol(panH,'Style','pushbutton',...
   'String','Process',...
   'Callback', {@ltpdv_fcn_run, 'spectral', panH, mainfig}, ...
   'Units', 'normalized', ...
   'Position',[pleft pbott pwidt phght]);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpdv_build_buildobject

PURPOSE ^

LTPDV_BUILD_BUILDOBJECT callback for the build object button on the build

SYNOPSIS ^

function ltpdv_build_buildobject(varargin)

DESCRIPTION ^

```
LTPDV_BUILD_BUILDOBJECT 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_BUILDOBJECT 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*

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 ^

```
% 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 $
%

function ltpdv_build_class_select(varargin)

  % Handles
  myh     = varargin{1};
  mainfig = varargin{end};

  % Fill in the set selector list
  sh = findobj(mainfig, 'Tag', 'LTPDVbuildSetSelect');

  % Get the selected class
  cl = ltpdv_build_get_selected_class(mainfig);

  % Get info object
  ii = eval([cl '.getInfo']);

  % Set sets
  set(sh, 'Value', 1);
  set(sh, 'String', ii.sets);

  % Callback
  ltpdv_build_set_select(sh, mainfig)

```

```
0031 end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_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

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   strs = get(sh, 'String');
0016   idx  = get(sh, 'Value');
0017   cl = strs{idx};
0018
0019 end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 $
%

function set = ltpdv_build_get_selected_pset(mainfig)

  % Fill in the set selector list
  sh = findobj(mainfig, 'Tag', 'LTPDVbuildSetSelect');

  % Get the selected class
  strs = get(sh, 'String');
  idx  = get(sh, 'Value');
  set = strs{idx};

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_fcnlist Handles
- ltpdv_preproc_sets_callback
- ltpdv_spectral_fcnlist Handles

SOURCE CODE ^

```
function ltpdv_build_params_panel(ph, ii, varargin)

  FONTSIZE = 12;

  pmarg = 0.025;

  keyl = pmarg;
  keyh = 0.055;
  keyw = 0.3;
  keyb = 1-pmarg-keyh;

  vall = pmarg*2 + keyw;
  valw = 0.5;

  al   = vall + valw + pmarg;
  aw   = 0.05;

  delete(get(ph, 'Children'))

  if numel(varargin) > 0
    activated = varargin{1};
    if strcmpi(activated, 'on')
      chkVal = 1;
    else
      chkVal = 0;
    end
  else
    activated = 'off';
    chkVal = 0;
  end

  pancomps = {};

  if isa(ii, 'minfo')
    % we give a pop-up menu of sets
```

```
    eh = uicontrol(ph,'Style','popupmenu',...
      'String',ii.sets,...
      'units', 'normalized', ...
      'BackgroundColor', 'w', ...
      'Fontsize', FONTSIZE, ...
      'Enable', 'on', ...
      'Position',[vall keyb valw keyh]);

    set(eh, 'Callback', {@ltpdv_preproc_sets_callback, ii});
  else
    pl = ii;

    if numel(pl.params) == 0
        sth = uicontrol(ph,'Style','text',...
          'String', 'contains no parameters',...
          'Units', 'normalized', ...
          'BackgroundColor', 'w', ...
          'Fontsize', FONTSIZE, ...
          'Position',[keyl keyb 2*keyw keyh]);
    end
    % check for any unsupported parameters
    unsupported = false;
    for j=1:numel(pl.params)
      val = pl.params(j).val;
      if isa(val, 'timespan')
        unsupported = true;
      end
    end
    if unsupported
        sth = uicontrol(ph,'Style','text',...
          'String', ['contains unsupported parameter of type: ' class(val)],...
          'Units', 'normalized', ...
          'BackgroundColor', 'w', ...
          'Fontsize', FONTSIZE, ...
          'Position',[keyl keyb 2*keyw keyh]);

    else
      % Build parameters
      for j=1:numel(pl.params)

        % Get key and val
        key = pl.params(j).key;
        val = pl.params(j).val;

        switch class(val)
          case 'char'
            valstr = val;
          case 'double'
            valstr = mat2str(val);
          case 'specwin'
            valstr = specwin.getTypes;
          otherwise
            valstr = char(val);
        end

        % key text
        sth = uicontrol(ph,'Style','text',...
          'String', key,...
          'Units', 'normalized', ...
          'BackgroundColor', 'w', ...
          'Fontsize', FONTSIZE, ...
          'Position',[keyl keyb keyw keyh]);

        % val edit
        if ischar(valstr) || isa(valstr, 'time') || isa(valstr, 'sym')
          if isa(valstr, 'time') || isa(valstr, 'sym')
            valstr = char(valstr);
          end
          eh = uicontrol(ph,'Style','edit',...
            'String', valstr,...
            'units', 'normalized', ...
            'BackgroundColor', 'w', ...
            'Fontsize', FONTSIZE, ...
            'Enable', activated, ...
            'Position',[vall keyb valw keyh]);
        elseif iscell(valstr)
          eh = uicontrol(ph,'Style','popupmenu',...
            'String',valstr,...
            'units', 'normalized', ...
            'BackgroundColor', 'w', ...
            'Fontsize', FONTSIZE, ...
            'Enable', activated, ...
            'Position',[vall keyb valw keyh]);
        else
          valstr
          error('### Unknown type for value string.');
```

```
        end
        setappdata(eh, 'valClass', class(val));
        % Activate
        ah = uicontrol(ph,'Style','checkbox',...
          'String','',...
          'units', 'normalized', ...
          'BackgroundColor', get(ph, 'BackgroundColor'), ...
          'Fontsize', FONTSIZE, ...
          'Value', chkVal, 'Position', [al keyb aw keyh], ...
          'Callback', {@ltpdv_preproc_param_act, ph});

        pancomps = [pancomps; {sth, eh, ah}];

        % Do next line
        keyb = keyb - pmarg - keyh;

      end
    end
  end

  setappdata(ph, 'pancomps', pancomps);


end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 $
%
function ltpdv_build_set_select(varargin)

  % Handles
  myh     = varargin{1};
  mainfig = varargin{end};

  % Get selected class
  cl = ltpdv_build_get_selected_class(mainfig);

  % Get selected set
  set = ltpdv_build_get_selected_pset(mainfig);

  % Get parameter set
  ii = eval([cl '.getInfo(''' cl ''',''' set ''')']);
  pl = ii.plists;

  % Build parameter list
  ph = findobj(mainfig, 'Tag', 'LTPDVbuildParamsPanel');
  ltpdv_build_params_panel(ph, pl, 'on')

```

```
0028 end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_calc_btn(varargin)
% 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 $
%

% Handles
myh      = varargin{1};
mainfig = varargin{end-1};
nins     = varargin{end};

% Get operation
op = get(myh, 'String');

% Get selected objects
objs = ltpdv_get_selected_objs(mainfig);
Nobjs = length(objs);

if Nobjs < nins
   error('### Please select two objects');
end

% check objects are all AOs
for j=1:Nobjs
   if ~isa(objs{j}, 'ao')
      error('### Please select only AOs for math calculations');
   end
end

% Loop over AOs
n   = 1;
```

```
res = {};
while n <= length(objs)
   % operate on two objects
   switch nins
      case 1
         res = [res {feval(op, objs{n})}];
      case 2
         if isempty(res)
            res = feval(op, objs{n}, objs{n+1});
            n = n + 1;
         else
            res = feval(op, res, objs{n});
         end
   end
   n = n + 1;
end

if nins == 2
   res = {res};
end

% Add this result to the object list
objs = getappdata(mainfig, 'LTPDAobjects');
if ~isempty(res)
   setappdata(mainfig, 'LTPDAobjects', [objs res]);
   % Refresh object list
   ltpdv_refresh_object_list(mainfig);
end


% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_cb_mainfigClose(varargin)

% 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 $
%

disp('* Goodbye from the LTPDA Dataviewer *')
delete(varargin{1})
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_delete_objects(varargin)

% 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 $
%

% Handles
clearAll = false;
if nargin == 3
   myh     = varargin{1};
   mainfig = varargin{end};
elseif nargin == 4
   myh     = varargin{1};
   mainfig = varargin{end-1};
   if strcmp(varargin{end}, 'all')
      clearAll = true;
   end
end

% Get selected rows
rows = getappdata(mainfig, 'ObjectListSelectedRows');

if clearAll
   % Set new object list
   setappdata(mainfig, 'LTPDAobjects', {});
else
   % Get current object list
   objs = getappdata(mainfig, 'LTPDAobjects');

   % Build index list to keep
   a   = 1:length(objs);
   idx = a(~ismember(a,rows));
   % Set new object list
   setappdata(mainfig, 'LTPDAobjects', objs(idx));
end

```

```
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*

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 ^

```
function ltpdv_explore_objects(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};


% Get selected objects
objs = ltpdv_get_selected_objs(mainfig);

% Pass to ltpda_explorer
ltpda_explorer(objs)

% 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 ^

```
function ltpdv_export_objects(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};


% check current workspace variables
basename = 'ex_obj';
ex = 0;
n  = [];
vars = evalin('base', 'who');
for j=1:length(vars)
  v = char(vars{j});
  if strncmp(basename, v, length(basename)) == 1
    % get the number
    n = [n str2num(v(end-1:end))];
  end
end

% Get selected objects
objs = ltpdv_get_selected_objs(mainfig);

% Loop over selected objects
for j=1:numel(objs)
  obj = objs{j};
  ex = max([n ex])+1;
  varname = sprintf('%s_%02d', basename, ex);
  assignin('base', varname, obj);
end


% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_fcn_run(varargin)
  % 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 $
  %

  % Get handles
  myh      = varargin{1};
  tag      = varargin{end-2};
  panH     = varargin{end-1};
  mainfig = varargin{end};

  % Get function
  switch tag
    case 'preproc'
      listtag = 'LTPDVpreprocfcnlist';
      ptag    = 'LTPDVpreprocparampan';
    case 'spectral'
      listtag = 'LTPDVspectralfcnlist';
      ptag    = 'LTPDVspectralparampan';
    otherwise
      error('### Unknown function set');
  end
  fcnh = findobj(panH, 'Tag', listtag);
  val  = get(fcnh, 'Value');
  strs = get(fcnh, 'String');
```

```
  fcn  = strs{val};

  % Get params
  parampanh = findobj(panH, 'Tag', ptag);
  pl = ltpdv_get_params_from_panel(parampanh);

  % Get selected objects
  objs = ltpdv_get_selected_objs(mainfig);

  % Collect AOs
  as = [];
  for j=1:numel(objs)
    obj = objs{j};
    if isa(obj, 'ao')
      as = [as obj];
    else
      warning('!!! Skipping object %s - it''s not an AO', obj.name);
    end
  end

  % Evaluate fcn
  res = feval(fcn, as, pl);

  % Add result to object list
  for j=1:numel(res)
    objs = getappdata(mainfig, 'LTPDAobjects');
    setappdata(mainfig, 'LTPDAobjects', [objs {res(j)}]);
  end
  % Refresh object list
  ltpdv_refresh_object_list(mainfig);

end

  % END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_filterAdd(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get existing filters
filts = getappdata(mainfig, 'Filters');

% Get current filter
filt = ltpdv_get_selected_filter(htab);

% Check name
count = 1;
while true
   found = false;
   for j=1:length(filts)
      if strcmp(filt.name, filts(j).name)
         if count > 1
            filt.setName(sprintf('%s%02d', filt.name(1:end-2), count), 'internal');
         else
            filt.setName(sprintf('%s%02d', filt.name, count), 'internal');
         end
         found = true;
         count = count + 1;
      end
   end
   if ~found
      break;
   end
end

% Add to list
setappdata(mainfig, 'Filters', [filts filt]);

% Refresh list
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 ^

```
function ltpdv_filterApply(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get selected filter
filts = getappdata(mainfig, 'Filters');
% Get selection
fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
idx = get(fh, 'Value');
sfilts = [];
for j=1:length(filts)
   if ismember(j, idx)
      sfilts = [sfilts filts(j)];
   end
end

%--------- Apply filter to selected objects

% Get selected objects
objs = ltpdv_get_selected_objs(mainfig);

% Collect AOs
as = [];
for j=1:numel(objs)
   obj = objs{j};
   if isa(obj, 'ao')
      as = [as obj];
   else
      warning('!!! Skipping object %s - it''s not an AO', obj.name);
   end
end

% Evaluate fcn
out = as;
```

```
for j=1:length(sfilts)
   filt = sfilts(j);
   out = feval('filter', out, plist('filter', filt));
end

% Add result to object list
for j=1:numel(out)
   objs = getappdata(mainfig, 'LTPDAobjects');
   setappdata(mainfig, 'LTPDAobjects', [objs {out(j)}]);
end
% Refresh object list
ltpdv_refresh_object_list(mainfig);


```

*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 ^

```
function ltpdv_filterBrowse(varargin)

% Handles
myh     = varargin{1};
panh    = varargin{end-1};
mainfig = varargin{end};

% File browser
[filename, pathname, filterindex] = uigetfile( ...
   {'*.fil', 'LISO Files'; '*.xml', 'LTPDA XML File'; '*.mat', 'MAT-File'}, 'Select Filter File');

if isequal(filename,0) || isequal(pathname,0)
else
   % Full file path
   fpath = fullfile(pathname, filename);

   % Set file name in file box
   fbh = findobj(panh, 'Tag', 'LTPDVfilterFileFilename');
   set(fbh, 'String', filename);

   % Load filter
   filt = miir(fpath);
   filt.name = filename;

   % Store filter
   setappdata(panh, 'filter', filt);

   % Filter info
   fih = findobj(panh, 'Tag', 'LTPDVfilterFileInfo');
   set(fih, 'String', display(filt));
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_filterDelete(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get selection
fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
idx = get(fh, 'Value');

% Get filters
filts = getappdata(mainfig, 'Filters');

fo = [];
for j=1:length(filts)
   if j~=idx
      fo = [fo filts(j)];
   end
end

setappdata(mainfig, 'Filters', fo);
set(fh, 'Value', idx-1);
% Refresh list
ltpdv_set_filter_list(mainfig);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_filterImport(varargin)

% Handles
myh     = varargin{1};
panh    = varargin{end-1};
mainfig = varargin{end};

% Get selected objects
filts = ltpdv_get_selected_objs(mainfig);
if length(filts) > 1
   error('### Select a single filter object');
end
filt = filts{1};
if ~isa(filt, 'miir')
   error('### Select a single miir filter object');
end

% Store filter
setappdata(panh, 'filter', filt);

% Filter info
fih = findobj(panh, 'Tag', 'LTPDVfilterObjectInfo');
set(fih, 'String', display(filt));

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function chansout = filter_control_chans(channels)

% 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 $
%

nchans = size(channels,1);
chansout = [];
for c=1:nchans
  idx = strfind(channels(c,:), '#');
  if isempty(idx)
    chansout = strvcat(chansout, deblank(channels(c,:)));
  end
end

chansout = char(sort(cellstr(chansout)));

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function filt = ltpdv_filter_from_standard_panel(panh)
% 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 $
%


% Filter type
h      = findobj(panh, 'Tag', 'LTPDVfiltStandardType');
idx    = get(h, 'Value');
strs   = get(h, 'String');
ftype = strs{idx};

disp(sprintf('** Building %s filter', ftype))

% Gain
h      = findobj(panh, 'Tag', 'LTPDVfiltStandardGain');
gain   = str2double(get(h, 'String'));

% Fc
h  = findobj(panh, 'Tag', 'LTPDVfiltStandardCutoff');
fc = str2num(get(h, 'String'));

% Fs
h  = findobj(panh, 'Tag', 'LTPDVfiltStandardFs');
fs = str2double(get(h, 'String'));

% Order
h      = findobj(panh, 'Tag', 'LTPDVfiltStandardOrder');
order = str2double(get(h, 'String'));

% Ripple
h       = findobj(panh, 'Tag', 'LTPDVfiltStandardRipple');
ripple = str2double(get(h, 'String'));

```

```
% Build plist
pl = plist('type', lower(ftype), ...
           'Gain', gain, ...
           'fc', fc, ...
           'fs', fs, ...
           'order', order, ...
           'ripple', ripple);

% Build filter
filt = miir(pl);

% 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 ^

```
function pzm = ltpdv_filter_get_pzmodel(mainfig)

% Get poles
poles = getappdata(mainfig, 'FilterPoles');

% Get zeros
zeros = getappdata(mainfig, 'FilterZeros');

% Get gain
gh = findobj(mainfig, 'Tag', 'LTPDVfilterPZgain');
gstr = get(gh, 'String');
if ~isempty(gstr)
   gain = num2str(gstr);
else
   gain = 1;
end

% Make model
pzm = pzmodel(gain, poles, zeros);

% Get name
nh = findobj(mainfig, 'Tag', 'LTPDVfilterPZname');
nstr = get(nh, 'String');
if ~isempty(nstr)
   pzm.name = nstr;
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function pl = ltpdv_filter_getf(panh)

% get f1
h = findobj(panh, 'Tag', 'LTPDVfilterF1');
f1 = str2num(get(h, 'String'));
% get f2
h = findobj(panh, 'Tag', 'LTPDVfilterF2');
f2 = str2num(get(h, 'String'));
% get nf
h = findobj(panh, 'Tag', 'LTPDVfilterNf');
nf = str2num(get(h, 'String'));
% get scale
h     = findobj(panh, 'Tag', 'LTPDVfilterScale');
idx   = get(h, 'Value');
strs = get(h, 'String');
scale = strs{idx};

pl = plist('scale', scale);
if ~isempty(f1)
   pl = append(pl, 'f1', f1);
end
if ~isempty(f2)
   pl = append(pl, 'f2', f2);
end
if ~isempty(nf)
   pl = append(pl, 'nf', nf);
end

% 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 ^

```
function ltpdv_filter_list_select(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get filters
filts = getappdata(mainfig, 'Filters');

% Get selected index
idx = get(myh, 'Value');

filts(idx)

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_filtersClear(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Clear list
setappdata(mainfig, 'Filters', []);

% Refresh list
ltpdv_set_filter_list(mainfig);


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpdv_getWorkspaceObjs

PURPOSE ^

LTPDV_GETWORKSPACEOBJS gets a list of LTPDA user objects currently in the

SYNOPSIS ^

function objs = ltpdv_getWorkspaceObjs()

DESCRIPTION ^

```
LTPDV_GETWORKSPACEOBJS 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 ^

```
function objs = ltpdv_getWorkspaceObjs()
% LTPDV_GETWORKSPACEOBJS 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 $
%

% get base workspace variables
ws_vars = evalin('base','whos');

objs = [];
for j=1:length(ws_vars)

   cmd = sprintf('obj = evalin(''base'', ''%s'');', ws_vars(j).name);
   eval(cmd)
   if isa(obj, 'ltpda_uo')
      objs = [objs ws_vars(j)];
   end

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_BUILDOBJECT callback for the build object button on the build
- ltpdv_fcn_run LTPDV_FCN_RUN runs the command on the selected panel

SOURCE CODE ^

```
% 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 $
%
function pl = ltpdv_get_params_from_panel(parampanh)


  % Get params
  pancomps  = getappdata(parampanh, 'pancomps');

  % loop over params
  pl = plist();
  for j=1:size(pancomps,1)

    % handles
    keyf = pancomps{j,1};
    valf = pancomps{j,2};
    actf = pancomps{j,3};

    % Get activate status
    if get(actf, 'Value')
      % Get key name
      key = get(keyf, 'String');

      % Get param value
      valclass = getappdata(valf, 'valClass');

      switch valclass
        case 'char'
          val = get(valf, 'String');
        case 'double'
          val = get(valf, 'String');
```

```
            try
              val = eval(val);
            end
          case 'sym'
            val = get(valf, 'String');
            val = sym(val);
          case 'specwin'
            strs = get(valf, 'String');
            idx  = get(valf, 'Value');
            val  = strs{idx};
            if strcmp(val, 'Kaiser')
              val = specwin(val, 10, 150);
            else
              val = specwin(val, 10);
            end
          case 'time'
            val = get(valf, 'String');
            val = time(val);
          otherwise
            error(['### Unsupported value class: ' valclass]);
        end

        % Add to plist
        pl = append(pl, key, val);
      end
    end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function filt = ltpdv_get_selected_filter(htab)


% Get selected panel
pidx = get(htab, 'SelectedIndex');
pans = get(htab, 'Children');
panh = pans(pidx);


% Get filter from panel handle
switch get(panh, 'Title')
   case 'Standard'
      % Build filter from standard panel
      filt = ltpdv_filter_from_standard_panel(panh);
   case 'File'
      filt = getappdata(panh, 'filter');
   case 'Object'
      filt = getappdata(panh, 'filter');
   case 'Poles/Zeros'
      mainfig = findobj(0, 'Tag', 'LTPDVmainfig');
      pzm = ltpdv_filter_get_pzmodel(mainfig);
      filt = miir(plist('pzmodel', pzm));
   otherwise
      error('### Unknown panel.');
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function objs = ltpdv_get_selected_objs(mainfig)

% 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 $
%

% Get selected rows
rows = getappdata(mainfig, 'ObjectListSelectedRows');
% Get current object list
objs = getappdata(mainfig, 'LTPDAobjects');
% Return required objects
objs = objs(rows);

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_hist_plot(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get selected objects
objs = ltpdv_get_selected_objs(mainfig);

% Pass AOs to iplot
for j=1:length(objs)
   obj = objs{j};
   if isa(obj, 'ao')
     plot(obj.hist);
   else
      warning('!!! Skipping object %s - it''s not an AO.', obj.name);
   end
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpdv_importObjs

PURPOSE ^

LTPDV_IMPORTOBJS import given objects to the object list

SYNOPSIS ^

function ltpdv_importObjs(varargin)

DESCRIPTION ^

```
LTPDV_IMPORTOBJS 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 ^

```
function ltpdv_importObjs(varargin)
% LTPDV_IMPORTOBJS 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 $
%

myh     = varargin{1};
mainfig = varargin{end-1};
oh      = varargin{end};

% Get data
data = get(oh, 'Data');

% Get selected rows
rows = getappdata(mainfig, 'ImportWorkspaceSelectedRows');

% Do we rename to variable name?
ch = findobj(mainfig, 'Tag', 'LTPDVusevarnameChk');
if get(ch, 'Value')
   useVarname = true;
else
   useVarname = false;
end

% Data to import
idata = data(rows,:);
% Current objects
% Get each object from the workspace
for j=1:size(idata,1)
   % get variable name
   varname = idata{j,1};
   % get object
   obj = evalin('base',  varname);
   if useVarname
```

```
0037        for k=1:numel(obj)
0038           obj(k) = obj(k).setName(varname);
0039        end
0040     end
0041     % Add these objecs 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*

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 ^

```
function ltpdv_load_objects(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get filename from user
[filename, pathname] = uigetfile( ...
  {'*.mat', 'MATLAB MAT Files (*.mat)'; ...
  '*.xml', 'LTPDA XML Files (*.xml)'}, ...
  'Select a file');

if isequal(filename,0) || isequal(pathname,0)
  disp('Cancelled.')
else
  fname = fullfile(pathname, filename);

  % split filename up
  [pathstr, name, ext, vers] = fileparts(fname);

  switch ext
    case '.mat'

      obj = load(fname);

    case '.xml'

      root_node = xmlread(filename);
      obj = utils.helper.xmlread(root_node);

    otherwise
      error('### Unsupported file format.');
  end
end

% Add this object to the list
objs = getappdata(mainfig, 'LTPDAobjects');
setappdata(mainfig, 'LTPDAobjects', [objs {obj}]);
% Refresh object list
ltpdv_refresh_object_list(mainfig);


```

```
% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_objectList_edit(varargin)
% 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 $
%

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Structure of selection details
sel = varargin{2};

% Get selected rows
rows = unique(sel.Indices(:,1));

% Get current object list
objs = getappdata(mainfig, 'LTPDAobjects');

objs{rows} = objs{rows}.setName(sel.NewData);

% Set objects back
setappdata(mainfig, 'LTPDAobjects', objs);

% Refresh object list
ltpdv_refresh_object_list(mainfig);


% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
0032     % get object
0033     obj = objs{rows};
0034
0035     ltpdv_properties_build_props(obj, mainfig)
0036
0037   end
0038
```

```

end

% END
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function control_channels = dv_parse_control_chans(channels)

% 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 $

```

```
% control data is sampled at 1Hz for now
count = 1;

numtoplevels = length(channels(:,1));

ch = 1;
control_channels = [];
for top=1:numtoplevels

  name = channels(top,:);
  if(length(name)>40)

    % parse the input string
    % get first part as top level
    [toplevel, r] = strtok(name, '#');
    name = r;
    while(length(name) > 1)

      % get sub level
      [sublevel, r] = strtok(name, '#');
      name = deblank(r);

      % get the number of sub sub levels
      [strN, r] = strtok(name, '#');
      name = deblank(r);
      % convert strN
      N = eval(strN);
      % now get each sub sub level making channel
      % names as we go and extracting data as we go
      for s = 1:N
        [subsublevel, r] = strtok(name, '#');
        name = deblank(r);
        control_channels = strvcat(control_channels,[toplevel ':' sublevel ':' subsublevel ]);
        ch = ch + 1;
      end
    end
  end
end

ch = ch-1;



% END

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_plot(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get selected objects
objs = ltpdv_get_selected_objs(mainfig);

% Pass AOs to iplot
as = [];
for j=1:length(objs)
   obj = objs{j}
   if isa(obj, 'ao')
      as = [as obj];
   else
      warning('!!! Skipping object %s - it''s not an AO.', obj.name);
   end
end

if ~isempty(as)
   iplot(as)
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_plotFilterResp(varargin)
% 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 $
%

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get selected filter
filts = getappdata(mainfig, 'Filters');
% Get selection
fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
idx = get(fh, 'Value');
sfilts = [];
for j=1:length(filts)
   if ismember(j, idx)
      sfilts = [sfilts filts(j)];
   end
end

% get frequency plist
pl = ltpdv_filter_getf(mainfig);

% Get responses
fr = [];
for j=1:length(sfilts)
   filt = sfilts(j);
   disp(sprintf('   - filter %03d: %s', j, filt.name));
   r = resp(filt, pl);
   if isempty(fr)
      fr = r;
```

```
    else
       fr = fr.*r;
    end
end

% Plot filter response
iplot(fr);

% 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 ^

```
function ltpdv_poleAdd(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};


% Get freq
fh = findobj(htab, 'Tag', 'LTPDVfilterPZfreq');
fstr = get(fh, 'String');
if ~isempty(fstr)
   f  = str2num(fstr);

   % Get Q
   qh = findobj(htab, 'Tag', 'LTPDVfilterPZq');
   q  = str2num(get(qh, 'String'));

   % Add to pole list
   poles = getappdata(mainfig, 'FilterPoles');
   if isempty(q)
      poles = [poles pz(f)];
   else
      poles = [poles pz(f,q)];
   end
   setappdata(mainfig, 'FilterPoles', poles);

   % Set pole list
   ltpdv_set_pole_list(mainfig);
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_poleClear(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Clear list
setappdata(mainfig, 'FilterPoles', []);

% Set pole list
ltpdv_set_pole_list(mainfig);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_poleDelete(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get selection
lh = findobj(htab, 'Tag', 'LTPDVfilterPolelist');
idx = get(lh, 'Value');

% Get poles
poles = getappdata(mainfig, 'FilterPoles');

% Make new list
ops = [];
for j=1:length(poles)
   if ~ismember(j, idx)
      ops = [ops poles(j)];
   end
end
setappdata(mainfig, 'FilterPoles', ops);

% Set pole list
ltpdv_set_pole_list(mainfig);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_preproc_fcnlist(varargin)

% Handles
myh     = varargin{1};
paramh  = varargin{end-1};
mainfig = varargin{end};

% Get the selected method
lidx = get(myh, 'Value');
lstr = get(myh, 'String');
fcn  = lstr{lidx};

% Get parameters for this method
ii = ao.getInfo(fcn);

% Build params panel
if numel(ii.plists) == 1
  ltpdv_build_params_panel(paramh, ii.plists);
else
  ltpdv_build_params_panel(paramh, ii);
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_preproc_param_act(varargin)


% Handles
myh     = varargin{1};
panh    = varargin{end};

% Parameter panel children handles
pancomps = getappdata(panh, 'pancomps');

% My state
val = get(myh, 'Value');

% My line
ml = find(myh==[pancomps{:,3}]);

if val
   set(pancomps{ml,2}, 'Enable', 'on');
else
   set(pancomps{ml,2}, 'Enable', 'off');
end



```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_preproc_sets_callback(varargin)

  myh = varargin{1};
  info = varargin{end};

  seti = get(myh, 'Value');

  plist = info.plists(seti);

  ltpdv_build_params_panel(get(myh, 'Parent'), plist);


end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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 $
%

function ltpdv_properties_apply(varargin)

  % Handles
  myh     = varargin{1};
  oldVals = varargin{end-2};
  obj     = varargin{end-1};
  mainfig = varargin{end};
  panh    = get(myh, 'Parent');

  % go through each label and edit box in order
  lhs = findobj(panh, 'Style', 'text');
  ehs = findobj(panh, 'Style', 'edit');

  for kk=numel(lhs):-1:1
    fcn = get(lhs(kk), 'String');
    val = get(ehs(kk), 'String');

    % check the value has changed
    ov = oldVals{numel(lhs)-kk+1};
    if ~strcmp(val, ov)
      feval(fcn, obj, val);
    end
  end

  % Refresh object list
  ltpdv_refresh_object_list(mainfig);

```

```
0036   display(obj)
0037 end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_properties_build_props(obj, mainfig)

  % Object class
  cl = class(obj);

  % Get setter methods
  ms = methods(cl);
  setters = ms(strmatch('set', ms));

  % Draw properties panel
  panh = findobj(mainfig, 'Tag', 'LTPDVobjectProperties');
  delete(get(panh, 'Children'))

  pmarg = 0.025;
  keyl = pmarg;
  keyh = 0.055;
  keyw = 0.3;
  keyb = 1-pmarg-keyh;
  vall = pmarg*2 + keyw;
  valw = 0.5;
  oldVals = {};
  for kk=1:numel(setters)

    sfcn = setters{kk};
    prop = lower(strrep(sfcn, 'set', ''));
    val = [];
    try
      val  = obj.(prop);
    end
    if isa(val, 'ltpda_obj')
      val = char(val);
    end
    if isa(val, 'sym')
      val = char(val);
    end
    if ~isnumeric(val)
      oldVals = [oldVals {val}];
      % Draw label
```

```
      sth = uicontrol(panh,'Style','text',...
        'String', sfcn,...
        'Units', 'normalized', ...
        'BackgroundColor', 'w', ...
        'Fontsize', 12, ...
        'Position',[keyl keyb keyw keyh]);

      % Draw edit box
      eh = uicontrol(panh,'Style','edit',...
        'String', val,...
        'units', 'normalized', ...
        'BackgroundColor', 'w', ...
        'Fontsize', 12, ...
        'Enable', 'on', ...
        'Position',[vall keyb valw keyh]);

      keyb = keyb - pmarg - keyh;
    end
  end
  % Draw apply button
  pbh = uicontrol(panh,'Style','pushbutton',...
    'String','Apply',...
    'Callback', {@ltpdv_properties_apply, oldVals, obj, mainfig}, ...
    'Units', 'normalized', ...
    'Fontsize', 12, ...
    'Position',[pmarg pmarg 0.15 0.05]);

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_pzmResp(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get pzmodel
pzm = ltpdv_filter_get_pzmodel(mainfig);

% Make response
iplot(resp(pzm))

```

*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 ^

```
function ltpdv_pzmStore(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get pzmodel
pzm = ltpdv_filter_get_pzmodel(mainfig);

% Add to objects
objs = getappdata(mainfig, 'LTPDAobjects');
objs = [objs {pzm}];
setappdata(mainfig, 'LTPDAobjects', objs);

% Refresh object list
ltpdv_refresh_object_list(mainfig);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_pzmStoreResp(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get pzmodel
pzm = ltpdv_filter_get_pzmodel(mainfig);

% Add to objects
objs = getappdata(mainfig, 'LTPDAobjects');
objs = [objs {resp(pzm)}];
setappdata(mainfig, 'LTPDAobjects', objs);

% Refresh object list
ltpdv_refresh_object_list(mainfig);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_IMPORTOBJS 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 ^

```
function ltpdv_refresh_object_list(mainfig)

% 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 $
%

objs = getappdata(mainfig, 'LTPDAobjects');

% Go through object array
data = {};
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*

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 ^

```
function ltpdv_save_objects(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get selected objects
objs = ltpdv_get_selected_objs(mainfig);

% Get filename from user
[filename, pathname] = uiputfile( ...
  {'*.mat','MAT-files (*.mat)'; ...
  '*.xml','LTPDA-XML-files (*.xml)'}, ...
  'Save as')


if isequal(filename,0) || isequal(pathname,0)
  disp('Cancelled.')
else
  fname = fullfile(pathname, filename);
  % split filename up
  [pathstr, name, ext, vers] = fileparts(fname);

  switch ext
    case '.mat'

      save(fname, 'objs');

    case '.xml'


      % Save each object
      for j=1:numel(objs)

        obj = objs{j};

        % filename
        fname = fullfile(pathstr, sprintf('%s_%03d%s', name, j, ext));

        % save object
        save(obj, fname)
```

```
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*

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 ^

```
function ltpdv_search_channels(varargin)

% 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 $

% Handles
myh = varargin{1};
mainfig = varargin{end};

% channels
ch = findobj(mainfig, 'Tag', 'LTPDVchannelList');
chans = get(ch, 'String');

nchans = size(chans,1);
chansout = [];

% First tokenize the search string
sh = findobj(mainfig, 'Tag', 'LTPDVchannelSearchEdit');
sstr = get(sh, 'String');
curr = upper(sstr);
if isempty(curr)
   [latest, chans] = ltpdv_server_get_latest_chans(mainfig);
   % Set channel list
   ltpdv_server_set_channel_list(mainfig, chans);
else

  [t, r] = strtok(curr);
  cstr = [];
  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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_addTime(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Add to times list
tlist = getappdata(mainfig, 'TimesList');

% Build this time
start = ltpdv_server_get_start_time(mainfig);
stop  = ltpdv_server_get_stop_time(mainfig);
comm  = ltpdv_server_get_comment(mainfig);


if start > 0 && stop > 0
   t = {start, stop, comm};

   tlist = [tlist; t];

   setappdata(mainfig, 'TimesList', tlist);

   ltpdv_server_set_timeslist(mainfig);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_channellist(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get selected channels
idx = get(myh, 'Value');
if length(idx) == 1

   % get channel
   strs = cellstr(get(myh, 'String'));
   channel = strs{idx};

   % Get one second of data
   start = ltpdv_server_get_start_time(mainfig);
   stop = ltpdv_server_get_stop_time(mainfig);
   % Get server details
   [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);

   switch dtype
      case 'frame data'
         pltype = 'FRAMES';
      case 'h(t) frames'
         pltype = 'HFRAMES';
      case 'raw data'
         pltype = 'RAW';
      case 'hour trends'
         pltype = 'HOUR';
      case 'day trends'
         pltype = 'DAY';
      case 'chour trends'
         pltype = 'CHOUR';
      case 'cday trends'
         pltype = 'CDAY';
```

```
    end

    if start < 0
       % get latest
       [start, chans] = ltpdv_server_get_latest_chans(mainfig);
    end

    % get AO
    a = geo2ao(plist('Type', pltype, 'SERVER', server, 'port', port, 'channels', channel,
...
                    'starttime', start, 'nsecs', 1, 'STAT', stat));

else

end

```

*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 ^

```
function ltpdv_server_clearTimes(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Clear list
setappdata(mainfig, 'TimesList', {});
ltpdv_server_set_timeslist(mainfig)
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_datatype_select(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get data type
idx  = get(myh, 'Value');
strs = get(myh, 'String');
dtype = strs{idx};

% port field
porth = findobj(mainfig, 'Tag', 'LTPDVserverPortEdit');
% rds field
rdsh = findobj(mainfig, 'Tag', 'LTPDVserverRdsEdit');
% cal field
calh = findobj(mainfig, 'Tag', 'LTPDVserverCalEdit');
% stat field
stath = findobj(mainfig, 'Tag', 'LTPDVserverStat');
% server select
ssh = findobj(mainfig, 'Tag', 'LTPDVserverServer');

switch dtype
   case 'frame data'
      set(porth, 'Enable', 'on');
      set(rdsh, 'Enable', 'off');
      set(calh, 'Enable', 'off');
      set(stath, 'Enable', 'off');
      set(ssh, 'Value', 1, 'String', {'localhost/9000', '130.75.117.159/9000',
'130.75.117.73/9000'});

   case 'h(t) frames'
      set(porth, 'Enable', 'on');
      set(rdsh, 'Enable', 'on');
      set(calh, 'Enable', 'on');
      set(stath, 'Enable', 'off');
      set(ssh, 'Value', 1, 'String', {'localhost/9000', '130.75.117.159/9008',
'130.75.117.73/9007', '130.75.117.73/9008'});

      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);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_deleteTimes(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_getChannels(varargin)

myh = varargin{1};
mainfig = varargin{end};

[latest, chans] = ltpdv_server_get_latest_chans(mainfig);
% Set channel list
ltpdv_server_set_channel_list(mainfig, chans);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_getLatest(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get server details
[dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);

switch dtype
   case 'frame data'
      latest = m2fserv(server, port, 'MIN');
   case 'raw data'
      latest = m2dserv(server);
   case 'h(t) frames'
      latest = m2fserv(server, port, 'HMIN');
   case 'hour trends'
      latest = m2fserv(server, port, 'HOUR');
   case 'day trends'
      latest = m2fserv(server, port, 'DAY');
   otherwise
      error('### Unknown data type');
end

% Set start and stop times
sh = findobj(mainfig, 'Tag', 'LTPDVserverStartEdit');
set(sh, 'String', GPS2UTC(latest));
sh = findobj(mainfig, 'Tag', 'LTPDVserverStopEdit');
set(sh, 'String', GPS2UTC(latest));
ltpdv_server_set_duration(mainfig, latest, latest)
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function chans = ltpdv_server_get_channels(mainfig)

% Get handle to list
ch = findobj(mainfig, 'Tag', 'LTPDVchannelList');
idx = get(ch, 'Value');
strs = cellstr(get(ch, 'String'));

chans = strs(idx);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function comment = ltpdv_server_get_comment(mainfig)

% Get handle
ch = findobj(mainfig, 'Tag', 'LTPDVserverCommentEdit');

comment = get(ch, 'String');



```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function a = ltpdv_server_get_data(mainfig)

% Get times
times = ltpdv_server_get_times(mainfig);

% Get server details
[dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);
spl = plist('Type', dtype, 'SERVER', server, 'port', port);
% Get selected channels
chans = ltpdv_server_get_channels(mainfig);

% Loop over times and channels
a = [];
for k=1:length(times)

   ts = times(k);
   start = char(ts.startT);
   nsecs = ts.endT-ts.startT+1;
   % Loop over channels
   for j=1:length(chans)
      chan = chans{j};
      disp(sprintf('=== Getting %s | %s [%ds]', chan, start, nsecs));
      na = geo2ao(spl, plist('channels', chan, 'starttime', start, 'nsecs', nsecs));
      a = [a na];
   end % End chans loop
end % End time loop

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function [latest, chans] = ltpdv_server_get_latest_chans(mainfig)

% Get server details
[dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig);

switch dtype
   case 'frame data'

      latest = m2fserv(server, port, 'MIN');
      chans  = m2fserv(server, port, latest, 'MIN');

   case 'raw data'

      latest = m2dserv(server);
      chans  = m2dserv(server, latest);

   case 'h(t) frames'

      latest = m2fserv(server, port, 'HMIN');
      chans  = m2fserv(server, port, latest, 'HMIN', cal, rds);

   case 'hour trends'

      latest = m2fserv(server, port, 'HOUR');
      chans  = m2fserv(server, port, latest, 'HOUR');

   case 'day trends'

      latest = m2fserv(server, port, 'DAY');
      chans  = m2fserv(server, port, latest, 'DAY');

   otherwise
      error('### Unknown data type');
```

```
0034 end
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function [dtype, server, port, rds, cal, stat] = ltpdv_server_get_server(mainfig)


% Get data type
sh = findobj(mainfig, 'Tag', 'LTPDVserverDataType');
idx = get(sh, 'Value');
strs = get(sh, 'String');
dtype = strs{idx};

% Get server
sh = findobj(mainfig, 'Tag', 'LTPDVserverServerEdit');
server = get(sh, 'String');

% Get port
sh = findobj(mainfig, 'Tag', 'LTPDVserverPortEdit');
port = str2num(get(sh, 'String'));

% Get rds
sh = findobj(mainfig, 'Tag', 'LTPDVserverRdsEdit');
rds = str2num(get(sh, 'String'));

% Get cal
sh = findobj(mainfig, 'Tag', 'LTPDVserverCalEdit');
cal = str2num(get(sh, 'String'));

% Get stat
sh = findobj(mainfig, 'Tag', 'LTPDVserverStat');
idx = get(sh, 'Value');
strs = get(sh, 'String');
stat = strs{idx};
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function gps = ltpdv_server_get_start_time(mainfig)

% Handle
sh = findobj(mainfig, 'Tag', 'LTPDVserverStartEdit');

% Get string
sstr = get(sh, 'String');


if sstr(1) == '+'
   % get start
   start = ltpdv_server_get_stop_time(mainfig);
   gps = start + str2num(sstr(2:end)) -1;
   set(sh, 'String', GPS2UTC(gps));
   return
end
if sstr(1) == '-'
   % get start
   start = ltpdv_server_get_stop_time(mainfig);
   gps = start - str2num(sstr(2:end)) +1;
   set(sh, 'String', GPS2UTC(gps));
   return
end

% Is this GPS?
if ~isempty(sstr)
   try
      gps = eval(sstr);
      utc = GPS2UTC(gps);
   catch
      utc = sstr;
      gps = UTC2GPS(utc);
```

```
    end
else
    gps = -1;
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function gps = ltpdv_server_get_stop_time(mainfig)

% Handle
sh = findobj(mainfig, 'Tag', 'LTPDVserverStopEdit');

% Get string
sstr = get(sh, 'String');

if sstr(1) == '+'
   % get start
   start = ltpdv_server_get_start_time(mainfig);
   gps = start + str2num(sstr(2:end)) -1;
   set(sh, 'String', GPS2UTC(gps));
   return
end
if sstr(1) == '-'
   % get start
   start = ltpdv_server_get_start_time(mainfig);
   gps = start - str2num(sstr(2:end)) +1;
   set(sh, 'String', GPS2UTC(gps));
   return
end

% Is this GPS?
if ~isempty(sstr)
   try
      gps = eval(sstr);
      utc = GPS2UTC(gps);
   catch
      utc = sstr;
      gps = UTC2GPS(utc);
   end
```

```
else
   gps = -1;
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function tss = ltpdv_server_get_times(mainfig)


% Do we have a list?
lh = findobj(mainfig, 'Tag', 'LTPDVtimesList');
tlist = getappdata(mainfig, 'TimesList');

if isempty(tlist)

   % Get a single time from the entry boxes
   start = ltpdv_server_get_start_time(mainfig);
   stop = ltpdv_server_get_stop_time(mainfig);

   if start < 0 || stop < 0
      error('### Please set a start and stop time or add some times to the list.');
   end

   % Get comment
   comm = ltpdv_server_get_comment(mainfig);

   % Make a timespan object
   tss = timespan(plist('start', GPS2UTC(start), 'end', GPS2UTC(stop)));
   if ~isempty(comm)
      tss.setName(comm);
   end

else

   % build a vector of timespan objects
   tss = [];
   for j=1:size(tlist,1)
      t = tlist(j,:);
      ts = timespan(plist('start', GPS2UTC(t{1}), 'end', GPS2UTC(t{2})));
      if isempty(t{3})
         ts.setName('None');
      else
         ts.setName(t{3});
```

```
        end
        tss = [tss ts];
    end


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 ^

```
function ltpdv_server_import(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

a = ltpdv_server_get_data(mainfig);

% Add objects to the object list

% Get current objects
objs = getappdata(mainfig, 'LTPDAobjects');

% Add these new ones
for j=1:numel(a)
   objs = [objs {a(j)}];
end

% Refresh object list
setappdata(mainfig, 'LTPDAobjects', objs);
ltpdv_refresh_object_list(mainfig);


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_queryServer(varargin)

% Handles
myh      = varargin{1};
mainfig = varargin{end};

[latest, chans] = ltpdv_server_get_latest_chans(mainfig);

% Set start and stop times
sh = findobj(mainfig, 'Tag', 'LTPDVserverStartEdit');
set(sh, 'String', GPS2UTC(latest));
sh = findobj(mainfig, 'Tag', 'LTPDVserverStopEdit');
set(sh, 'String', GPS2UTC(latest));
ltpdv_server_set_duration(mainfig, latest, latest)

% Set channel list
ltpdv_server_set_channel_list(mainfig, chans);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_server_select(varargin)

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Get selected server
sh   = findobj(mainfig, 'Tag', 'LTPDVserverServer');
idx  = get(sh, 'Value');
strs = get(sh, 'String');
srv  = strs{idx};

% split port off
[server, port] = strtok(srv, '/');
if ~isempty(port)
   port = str2num(port(2:end));
end

% Set server edit box
eh = findobj(mainfig, 'Tag', 'LTPDVserverServerEdit');
set(eh, 'String', server);

% Set port edit box
eh = findobj(mainfig, 'Tag', 'LTPDVserverPortEdit');
set(eh, 'String', port);


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_set_channel_list(mainfig, chans)


% Get normal channels
channels = ltpdv_filter_control_chans(chans);

% Check is control checkbox
ch = findobj(mainfig, 'Tag', 'LTPDVchannelIncludeControl');
if get(ch, 'Value')
   % remove control chans
   control_chans = ltpdv_parse_control_chans(chans);
   channels = strvcat(control_chans, channels);
end

% Get channel list
ch = findobj(mainfig, 'Tag', 'LTPDVchannelList');
set(ch, 'Value', 1);
set(ch, 'String', channels);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_set_duration(mainfig, start, stop)

% Get handle
dh = findobj(mainfig, 'Tag', 'LTPDVserverDurationDisplay');

% Get duration
dur = stop - start + 1;

if dur < 0
  error('### The duration is negative. Change the start or stop time.');
end

hours = fix(dur/3600);
mins  = fix((dur-hours*3600)/60);
secs  = dur-hours*3600-mins*60;

sign = '';
if hours < 0
   sign = '-';
   hours = abs(hours);
end
if mins < 0
   sign = '-';
   mins = abs(mins);
end
if secs < 0
   sign = '-';
   secs = abs(secs);
end

set(dh, 'String', sprintf('%s%02d:%02d:%02d', sign, hours, mins, secs));


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_set_timeslist(mainfig)

% Get times list
tlist = getappdata(mainfig, 'TimesList');

% Get list handle
lh = findobj(mainfig, 'Tag', 'LTPDVtimesList');

% Format list
lstr = [];
for j=1:size(tlist,1)
   t = tlist(j,:)
   lstr = [lstr {sprintf('%s [%ds] / %s', GPS2UTC(t{1}), t{2}-t{1}+1, t{3})}];
end

set(lh, 'Value', 1, 'String', lstr);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_start_edit(varargin)

% Handles
myh      = varargin{1};
mainfig = varargin{end};

% Get start time
start = ltpdv_server_get_start_time(mainfig);
% Get stop time
stop = ltpdv_server_get_stop_time(mainfig);

ltpdv_server_set_duration(mainfig, start, stop);

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_stop_edit(varargin)

% Handles
myh      = varargin{1};
mainfig = varargin{end};

% Get start time
start = ltpdv_server_get_start_time(mainfig);
% Get stop time
stop = ltpdv_server_get_stop_time(mainfig);


ltpdv_server_set_duration(mainfig, start, stop);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_server_timeslist(varargin)
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_set_filter_list(mainfig)

% Get filters
filts = getappdata(mainfig, 'Filters');

% Get handle to filter list
fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');

data = {};
for j=1:length(filts)
   filt = filts(j);
   fc = find(filt.hist.plistUsed, 'fc');
   if ~isempty(fc)
      extra = mat2str(fc);
   else
      extra = '';
   end
   if isempty(extra)
      data = [data; {sprintf('%s', filt.name)}];
   else
      data = [data; {sprintf('%s | %s', filt.name, extra)}];
   end
end

set(fh, 'String', data)
set(fh, 'Value', 1)

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_set_pole_list(mainfig)

% Get poles
poles = getappdata(mainfig, 'FilterPoles');

% Get handle to pole list
lh = findobj(mainfig, 'Tag', 'LTPDVfilterPolelist');
set(lh, 'Value', 1);

% Build string
str = {};
for j=1:length(poles)
   p = poles(j);
   if isnan(p.q)
      str = [str {num2str(p.f)}];
   else
      str = [str {[num2str(p.f) ', ' num2str(p.q)]}];
   end
end

set(lh, 'String', str);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_set_zero_list(mainfig)

% Get poles
poles = getappdata(mainfig, 'FilterZeros');

% Get handle to pole list
lh = findobj(mainfig, 'Tag', 'LTPDVfilterZerolist');
set(lh, 'Value', 1);

% Build string
str = {};
for j=1:length(poles)
   p = poles(j);
   if isnan(p.q)
      str = [str {num2str(p.f)}];
   else
      str = [str {[num2str(p.f) ', ' num2str(p.q)]}];
   end
end

set(lh, 'String', str);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_spectral_fcnlist(varargin)

% Handles
myh     = varargin{1};
paramh  = varargin{end-1};
mainfig = varargin{end};

% Get the selected method
lidx = get(myh, 'Value');
lstr = get(myh, 'String');
fcn  = lstr{lidx};

% Get parameters for this method
if ismethod(ao, fcn)
  ii = ao.getInfo(fcn);
  pl = ii.plists(1);
else
  pl = feval(fcn, 'Params');
end

% Build params panel
ltpdv_build_params_panel(paramh, pl);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_storeFilter(varargin)
% 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 $
%

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

filts = getappdata(mainfig, 'Filters');

% Get selection
fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
idx = get(fh, 'Value');
sfilts = [];
for j=1:length(filts)
   if ismember(j, idx)
      sfilts = [sfilts filts(j)];
   end
end

% Add filter to the object list
objs = getappdata(mainfig, 'LTPDAobjects');
for j=1:length(sfilts)
   filt = sfilts(j);
   objs = [objs {filt}];
end
setappdata(mainfig, 'LTPDAobjects', objs);

% Refresh object list
```

```
ltpdv_refresh_object_list(mainfig);


% END
```

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_storeFilterResp(varargin)

% 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 $
%

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get selected filter
filts = getappdata(mainfig, 'Filters');
% Get selection
fh = findobj(mainfig, 'Tag', 'LTPDVfilterList');
idx = get(fh, 'Value');
sfilts = [];
for j=1:length(filts)
   if ismember(j, idx)
      sfilts = [sfilts filts(j)];
   end
end

% get frequency plist
pl = ltpdv_filter_getf(mainfig);

% Get responses
fr = [];
for j=1:length(sfilts)
```

```
    filt = sfilts(j);
    disp(sprintf('    - filter %03d: %s', j, filt.name));
    r = resp(filt, pl);
    if isempty(fr)
       fr = r;
    else
       fr = fr.*r;
    end
end

% Add filter to the object list
objs = getappdata(mainfig, 'LTPDAobjects');
setappdata(mainfig, 'LTPDAobjects', [objs {fr}]);

% Refresh object list
ltpdv_refresh_object_list(mainfig);



% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

ltpdv_tabChanged

PURPOSE ^

SYNOPSIS ^

function ltpdv_tabChanged(varargin)

DESCRIPTION ^

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

- ltpdv_buildImportPanel

SOURCE CODE ^

```
function ltpdv_tabChanged(varargin)

myh   = varargin{1};
edata = varargin{2};
% get old tab
ti = edata.OldValue;
tabs = get(myh, 'Children');
if ti > 0
  oth  = tabs(ti);

  % Find all java objects below oth
  jobjs = findobj(oth, 'Type','uitabgroup');
  set(jobjs, 'Visible', 'off')

end

if ti > 0
  % get new tab
  ti = edata.NewValue;
  tabs = get(myh, 'Children');
  nth  = tabs(ti);

  % Find all java objects below oth
  jobjs = findobj(nth, 'Type','uitabgroup');
  set(jobjs, 'Visible', 'on');
end
drawnow

h = findobj(myh, 'Tag', 'LTPDVimportTabPanel');
if h~=myh
  set(h, 'SelectedIndex', 1);
end

end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_workspaceList_select(varargin)
% 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 $
%

% Handles
myh     = varargin{1};
mainfig = varargin{end};

% Structure of selection details
sel = varargin{2};

% Get selected rows
rows = unique(sel.Indices(:,1));
setappdata(mainfig, 'ImportWorkspaceSelectedRows', rows);

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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_GETWORKSPACEOBJS gets a list of LTPDA user objects currently in the

This function is called by:

- ltpdv_buildImportPanel

SOURCE CODE ^

```
function ltpdv_workspaceList_set_data(varargin)

% 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 $
%

% Get panel handle
th = varargin{end};

% Make a list of LTPDA workspace user objects
objs = ltpdv_getWorkspaceObjs;

% Make a data cell array
data = {};
for j=1:length(objs)
    data = [data; {objs(j).name, objs(j).class, mat2str(objs(j).size), objs(j).bytes/1024}];
end
% Set the data
set(th, 'Data', data);

% END
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_zeroAdd(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};


% Get freq
fh = findobj(htab, 'Tag', 'LTPDVfilterPZfreq');
fstr = get(fh, 'String');
if ~isempty(fstr)
   f  = str2num(fstr);

   % Get Q
   qh = findobj(htab, 'Tag', 'LTPDVfilterPZq');
   q  = str2num(get(qh, 'String'));

   % Add to zero list
   zeros = getappdata(mainfig, 'FilterZeros');
   if isempty(q)
      zeros = [zeros pz(f)];
   else
      zeros = [zeros pz(f,q)];
   end
   setappdata(mainfig, 'FilterZeros', zeros);

   % Set pole list
   ltpdv_set_zero_list(mainfig);
end


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_zeroClear(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Clear list
setappdata(mainfig, 'FilterZeros', []);

% Set pole list
ltpdv_set_zero_list(mainfig);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpdv_zeroDelete(varargin)

% Handles
myh     = varargin{1};
htab    = varargin{end-1};
mainfig = varargin{end};

% Get selection
lh = findobj(htab, 'Tag', 'LTPDVfilterZerolist');
idx = get(lh, 'Value');

% Get zeros
zeros = getappdata(mainfig, 'FilterZeros');

% Make new list
ops = [];
for j=1:length(zeros)
   if ~ismember(j, idx)
      ops = [ops zeros(j)];
   end
end
setappdata(mainfig, 'FilterZeros', ops);

% Set pole list
ltpdv_set_zero_list(mainfig);
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%%% path: m/gui/pzmodel_designer %%%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%   path: m/gui/pzmodel_designer   %%%%%%%%%%%%%%%%%%

   <a href="matlab:help
m/gui/pzmodel_designer/pzmodel_helper">m/gui/pzmodel_designer/pzmodel_helper</a> -  M-file
for pzmodel_helper.fig
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = pzmodel_helper(varargin)
% 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 $
%


% Edit the above text to modify the response to help pzmodel_helper

% Last Modified by GUIDE v2.5 19-Apr-2007 10:37:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @pzmodel_helper_OpeningFcn, ...
                   'gui_OutputFcn',  @pzmodel_helper_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before pzmodel_helper is made visible.
function pzmodel_helper_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to pzmodel_helper (see VARARGIN)

% Choose default command line output for pzmodel_helper
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes pzmodel_helper wait for user response (see UIRESUME)
% uiwait(handles.main);

% create empty lists
poles = [];
zeros = [];

setappdata(handles.main, 'poles', poles);
setappdata(handles.main, 'zeros', zeros);
setappdata(handles.main, 'filt', []);

% --- Outputs from this function are returned to the command line.
function varargout = pzmodel_helper_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function pzEdit_Callback(hObject, eventdata, handles)
% hObject    handle to pzEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pzEdit as text
%        str2double(get(hObject,'String')) returns contents of pzEdit as a double


% --- Executes during object creation, after setting all properties.
function pzEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pzEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in addPoleBtn.
function addPoleBtn_Callback(hObject, eventdata, handles)
% hObject    handle to addPoleBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get values from pzEdit
strVal = get(handles.pzEdit, 'String');
val    = str2num(strVal);
f      = val(1);
if length(val) == 2
  q = val(2);
  p = pz(f,q);
else
  p = pz(f);
end

poles = getappdata(handles.main, 'poles');
poles = [poles p];
setappdata(handles.main, 'poles', poles);

% update list
updatePolesList(handles);

% update model
updateModelResp(handles);


% --- Executes on selection change in poleList.
function poleList_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to poleList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns poleList contents as cell array
%        contents{get(hObject,'Value')} returns selected item from poleList


% --- Executes during object creation, after setting all properties.
function poleList_CreateFcn(hObject, eventdata, handles)
% hObject    handle to poleList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in addZeroBtn.
function addZeroBtn_Callback(hObject, eventdata, handles)
% hObject    handle to addZeroBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get values from pzEdit
strVal = get(handles.pzEdit, 'String');
val    = str2num(strVal);
f      = val(1);
if length(val) == 2
  q = val(2);
  z = pz(f,q);
else
  z = pz(f);
end

zeros = getappdata(handles.main, 'zeros');
zeros = [zeros z];
setappdata(handles.main, 'zeros', zeros);

% update list
updateZerosList(handles);

% update model
updateModelResp(handles);


% --- Executes on selection change in zeroList.
function zeroList_Callback(hObject, eventdata, handles)
% hObject    handle to zeroList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns zeroList contents as cell array
%        contents{get(hObject,'Value')} returns selected item from zeroList


% --- Executes during object creation, after setting all properties.
function zeroList_CreateFcn(hObject, eventdata, handles)
% hObject    handle to zeroList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%----------- Update Poles List -----------------------------
function updatePolesList(handles)

poles = getappdata(handles.main, 'poles');
pstr = [];
for n=1:length(poles)
  p = poles(n);
  f = p.f;
  q = p.q;
  if q>0.5
    pstr = strvcat(pstr, sprintf('%2.2f Hz   Q=%2.2f', f, q));
  else
    pstr = strvcat(pstr, sprintf('%2.2f Hz', f));
```

```
  end
end

if isempty(poles)
  set(handles.poleList, 'String', '  ');
else
  set(handles.poleList, 'String', pstr);
end
set(handles.poleList, 'Value', 1);

%----------- Update Zeros List -----------------------------
function updateZerosList(handles)

zeros = getappdata(handles.main, 'zeros');
pstr = [];
for n=1:length(zeros)
  p = zeros(n);
  f = p.f;
  q = p.q;
  if q>0.5
    pstr = strvcat(pstr, sprintf('%2.2f Hz   Q=%2.2f', f, q));
  else
    pstr = strvcat(pstr, sprintf('%2.2f Hz', f));
  end
end

if isempty(zeros)
  set(handles.zeroList, 'String', '  ');
else
  set(handles.zeroList, 'String', pstr);
end
set(handles.zeroList, 'Value', 1);

%----------- Update Model response -----------------------------
function updateModelResp(handles)

gain = str2double(get(handles.gainEdit, 'String'));
poles = getappdata(handles.main, 'poles');
zeros = getappdata(handles.main, 'zeros');

pzm = pzmodel(gain, poles, zeros);

f1s = get(handles.f1Edit, 'String');
f2s = get(handles.f2Edit, 'String');
nfs = get(handles.nfEdit, 'String');
fss = get(handles.fsEdit, 'String');

if isempty(f1s)
  f1 = getlowerFreq(pzm)/10;
else
  f1 = str2double(f1s);
end
if isempty(f2s)
  f2 = getupperFreq(pzm)*10;
else
  f2 = str2double(f2s);
end
if isempty(nfs)
  nf = 1000;
else
  nf = str2double(nfs);
end

if ~isempty(pzm.poles) || ~isempty(pzm.zeros)

  % make model response
  a  = resp(pzm, f1, f2, nf);
  d  = a.data;
  mx = d.x;
  my = d.y;

  % make IIR response
  if isempty(fss)
    filt     = miir(plist([param('pzmodel', pzm)]));
  else
    filt     = miir(plist([param('pzmodel', pzm) param('fs', str2double(fss))]));
  end
  setappdata(handles.main, 'filt', filt);
  set(handles.filterStrEdit, 'String', string(filt));

  filtresp = resp(filt, plist(param('f', mx.')));
  d        = filtresp.data;
  fx       = d.x;
  fy       = d.y;

  axes(handles.magAxes) % Select the proper axes
```

```
  loglog(mx,abs(my), fx, abs(fy), 'r--')
  grid on;
  axis tight
  xlabel('');
  ylabel('Magnitude');
  legend('Pole/Zero model', 'IIR','Location','SouthWest')

  axes(handles.phaseAxes) % Select the proper axes
  semilogx(mx, utils.math.phase(my), fx, utils.math.phase(fy), 'r--')
  grid on;
  axis tight
  xlabel('Frequency [Hz]');
  ylabel('Phase [deg]');
else
  axes(handles.magAxes) % Select the proper axes
  cla
  axes(handles.phaseAxes) % Select the proper axes
  cla
end



function gainEdit_Callback(hObject, eventdata, handles)
% hObject    handle to gainEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gainEdit as text
%        str2double(get(hObject,'String')) returns contents of gainEdit as a double

updateModelResp(handles)

% --- Executes during object creation, after setting all properties.
function gainEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gainEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function f1Edit_Callback(hObject, eventdata, handles)
% hObject    handle to f1Edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of f1Edit as text
%        str2double(get(hObject,'String')) returns contents of f1Edit as a double


% --- Executes during object creation, after setting all properties.
function f1Edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to f1Edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function f2Edit_Callback(hObject, eventdata, handles)
% hObject    handle to f2Edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of f2Edit as text
%        str2double(get(hObject,'String')) returns contents of f2Edit as a double


% --- Executes during object creation, after setting all properties.
function f2Edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to f2Edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function nfEdit_Callback(hObject, eventdata, handles)
% hObject    handle to nfEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of nfEdit as text
%        str2double(get(hObject,'String')) returns contents of nfEdit as a double


% --- Executes during object creation, after setting all properties.
function nfEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nfEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in replotBtn.
function replotBtn_Callback(hObject, eventdata, handles)
% hObject    handle to replotBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

updateModelResp(handles)



function fsEdit_Callback(hObject, eventdata, handles)
% hObject    handle to fsEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of fsEdit as text
%        str2double(get(hObject,'String')) returns contents of fsEdit as a double
updateModelResp(handles)


% --- Executes during object creation, after setting all properties.
function fsEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fsEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in figureBtn.
function figureBtn_Callback(hObject, eventdata, handles)
% hObject    handle to figureBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

gain = str2double(get(handles.gainEdit, 'String'));
poles = getappdata(handles.main, 'poles');
zeros = getappdata(handles.main, 'zeros');

pzm = pzmodel(gain, poles, zeros);

f1s = get(handles.f1Edit, 'String');
f2s = get(handles.f2Edit, 'String');
nfs = get(handles.nfEdit, 'String');
fss = get(handles.fsEdit, 'String');

if isempty(f1s)
  f1 = getlowerFreq(pzm)/10;
```

```
else
  f1 = str2double(f1s);
end
if isempty(f2s)
  f2 = getupperFreq(pzm)*10;
else
  f2 = str2double(f2s);
end
if isempty(nfs)
  nf = 1000;
else
  nf = str2double(nfs);
end

% make model response
a  = resp(pzm, f1, f2, nf);
a.setName('PZmodel');

% make IIR response
if isempty(fss)
  filt     = miir(plist([param('pzmodel', pzm)]));
else
  filt     = miir(plist([param('pzmodel', pzm) param('fs', str2double(fss))]));
end


filtresp = resp(filt, plist(param('f', a.data.getX)));
filtresp.setName('IIR filter');

iplot(a, filtresp)



function filterStrEdit_Callback(hObject, eventdata, handles)
% hObject    handle to filterStrEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of filterStrEdit as text
%        str2double(get(hObject,'String')) returns contents of filterStrEdit as a double


% --- Executes during object creation, after setting all properties.
function filterStrEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to filterStrEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in clearBtn.
function clearBtn_Callback(hObject, eventdata, handles)
% hObject    handle to clearBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

setappdata(handles.main, 'poles', []);
setappdata(handles.main, 'zeros', []);

% update list
updateZerosList(handles);
updatePolesList(handles);

% update model
updateModelResp(handles);


% --- Executes on button press in deletePolesBtn.
function deletePolesBtn_Callback(hObject, eventdata, handles)
% hObject    handle to deletePolesBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get selection from list
values = get(handles.poleList, 'Value');

% get poles
poles = getappdata(handles.main, 'poles');

po = [];
```

```
for j=1:length(poles)
  if j~=values
    po = [po poles(j)];
  end
end

setappdata(handles.main, 'poles', po);
updatePolesList(handles);

% update model
updateModelResp(handles);


% --- Executes on button press in deleteZerosBtn.
function deleteZerosBtn_Callback(hObject, eventdata, handles)
% hObject    handle to deleteZerosBtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get selection from list
values = get(handles.zeroList, 'Value');
% get zeros
zeros = getappdata(handles.main, 'zeros');
zo = [];
for j=1:length(zeros)
  if j~=values
    zo = [zo zeros(j)];
  end
end

setappdata(handles.main, 'zeros', zo);
updateZerosList(handles);

% update model
updateModelResp(handles);


% --- Executes on button press in saveAObtn.
function saveAObtn_Callback(hObject, eventdata, handles)
% hObject    handle to saveAObtn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get response
gain = str2double(get(handles.gainEdit, 'String'));
poles = getappdata(handles.main, 'poles');
zeros = getappdata(handles.main, 'zeros');
pzm = pzmodel(gain, poles, zeros);

f1s = get(handles.f1Edit, 'String');
f2s = get(handles.f2Edit, 'String');
nfs = get(handles.nfEdit, 'String');
fss = get(handles.fsEdit, 'String');

if isempty(f1s)
  f1 = getlowerFreq(pzm)/10;
else
  f1 = str2double(f1s);
end
if isempty(f2s)
  f2 = getupperFreq(pzm)*10;
else
  f2 = str2double(f2s);
end
if isempty(nfs)
  nf = 1000;
else
  nf = str2double(nfs);
end

% make model response
a  = resp(pzm, f1, f2, nf);

% Get filename
[filename, pathname] = uiputfile('*.xml', 'LTPDA XML file (*.xml)', 'Save as');
if isequal(filename,0) || isequal(pathname,0)
else
  fname = fullfile(pathname, filename);
  save(a, fname);
end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = ltpdaquicklook(varargin)

% 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
```

```
0016 %
0017 %%%%
```

```
  'String','',...
  'BackgroundColor', 'w', ...
  'ForegroundColor', 'b', ...
  'HorizontalAlignment', 'left', ...
  'Tag', 'LTPDA_quicklook_display', ...
  'Position',[170 90 420 300]);

% Plot button
pbh = uicontrol(mainfig,'Style','pushbutton',...
  'String','iplot',...
  'Callback', {@call_iplot, mainfig}, ...
  'Position',[10 10 60 25]);

% history plot
pbh = uicontrol(mainfig,'Style','pushbutton',...
  'String','plot history',...
  'Callback', {@call_histplot, mainfig}, ...
  'Position',[80 10 80 25]);


%% Start the GUI

% Make the GUI visible.
set(mainfig,'Visible','on')

%% Callbacks

%---------------- history plot
function call_histplot(varargin)

mainfig = varargin{end};
objs = getappdata(mainfig, 'objs');

% get selection
olh = findobj(mainfig, 'Tag', 'LTPDA_quicklook_objlist');
idx = get(olh, 'Value');

cmd = sprintf('obj = evalin(''base'', ''%s'');', objs(idx).name);
eval(cmd);

plot(obj.hist);


%---------------- iplot
function call_iplot(varargin)

mainfig = varargin{end};
objs = getappdata(mainfig, 'objs');

% get selection
olh = findobj(mainfig, 'Tag', 'LTPDA_quicklook_objlist');
idx = get(olh, 'Value');

cmd = sprintf('obj = evalin(''base'', ''%s'');', objs(idx).name);
eval(cmd);

iplot(obj);

%---------------- refresh
function refresh(varargin)

mainfig = varargin{end};
objs = getWorkspaceObjs();
setWorkspaceObjsList(objs)
setappdata(mainfig, 'objs', objs);

%---------------- Close function
function ltpda_quicklook_close(varargin)
% Callback executed when the GUI is closed

disp('* Goodbye from the LTPDA Quicklook GUI *')
delete(varargin{1})


%---------------- Get a list of LTPDA objects in the MATLAB workspace
function objs = getWorkspaceObjs()

% get base workspace variables
ws_vars = evalin('base','whos');

objs = [];
for j=1:length(ws_vars)
  cmd = sprintf('obj = evalin(''base'', ''%s'');', ws_vars(j).name);
  eval(cmd)
  if isa(obj, 'ltpda_uo')
    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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%%% path: m/gui/specwin_viewer %%%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   path: m/gui/specwin_viewer   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help
m/gui/specwin_viewer/ltpda_specwin_viewer_build_window">m/gui/specwin_viewer/ltpda_specwin_viewe
- Function to build the window and display it.
   <a href="matlab:help
m/gui/specwin_viewer/ltpda_specwin_viewer_close">m/gui/specwin_viewer/ltpda_specwin_viewer_close
- Callback executed when the GUI is closed
   <a href="matlab:help
m/gui/specwin_viewer/ltpda_specwin_viewer_wintype">m/gui/specwin_viewer/ltpda_specwin_viewer_win
- Callback executed when the user selects a window
   <a href="matlab:help
m/gui/specwin_viewer/specwin_viewer">m/gui/specwin_viewer/specwin_viewer</a>
-  allows the user to explore spectral windows.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_specwin_viewer_build_window(varargin)
% Function to build the window and display it.

plotType = varargin{end};

mainfig = findobj('Tag', 'LTPDAspecwin_viewer');
ax      = getappdata(mainfig, 'axes');
info    = findobj('Tag', 'LTPDA_specwin_viewer_wininfo');

% get window type
winTxt = findobj('Tag', 'LTPDA_specwin_viewer_WinType');
wins   = get(winTxt, 'String');
win    = wins{get(winTxt, 'Value')};

% Get window size
sizeTxt = findobj('Tag', 'LTPDA_specwin_viewer_WinSize');
winlen  = str2num(get(sizeTxt, 'String'));

if strcmpi(win, 'Kaiser')
  % get psll
  psllTxt = findobj('Tag', 'LTPDA_specwin_viewer_PSLL');
  psll    = str2num(get(psllTxt, 'String'));
  w = specwin(win, winlen, psll);
  cstr = sprintf('specwin(''%s'', %d, %f)', win, winlen, psll);
else
  w = specwin(win, winlen);
  cstr = sprintf('specwin(''%s'', %d)', win, winlen);
end

consTxt = findobj(mainfig, 'Tag', 'LTPDA_specwin_viewer_cstr');
set(consTxt, 'String', cstr);


titleStr  = '';
legendStr = '';

switch plotType
  case 'Time-domain'
    hl = plot(ax, 1:length(w.win), w.win);
    xlabel('sample');
    ylabel('amplitude');
  case 'Freq-domain'
    % Freq response
```

```
    f = linspace(-30,30,1000);
    r = zeros(size(f));
    N = length(w.win);

    % do dft
    for j=1:length(f)
      k = [0:N-1].';
      r(j) = (w.win * exp(-2*pi*1i*f(j)*k/N) )./w.ws;
    end

    % convert to dB
    dbr = 20*log10(abs(r));    hl = plot(ax, f, dbr);
    xlabel('bin');
    ylabel('amplitude [dB]');
end

col = [0.8 0.1 0.1];
set(hl, 'Color', col);
titleStr = [titleStr, utils.plottools.label(w.type), ', '];
titleStr = titleStr(1:end-2);
title(sprintf('Window: %s', titleStr));
axis tight
grid on


% Info string
lstr = [sprintf('alpha = %g\n', w.alpha)...
  sprintf('psll = %g\n', w.psll)...
  sprintf('rov = %g\n', w.rov)...
  sprintf('nenbw = %g\n', w.nenbw)...
  sprintf('w3db = %g\n', w.w3db)...
  sprintf('flatness = %g\n', w.flatness)];
legendStr = [legendStr cellstr(lstr)];
set(info, 'String', legendStr);

% Constructor
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_specwin_viewer_close(varargin)
% Callback executed when the GUI is closed

disp('* Goodbye from the LTPDA Spectral Window Viewer *')
delete(varargin{1})

end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function ltpda_specwin_viewer_wintype(varargin)
% Callback executed when the user selects a window


etxt = findobj('Tag', 'LTPDA_specwin_viewer_PSLL');
wsel = findobj('Tag', 'LTPDA_specwin_viewer_WinType');

idx  = get(wsel, 'Value');
wins = get(wsel, 'String');
win  = wins{idx};


if strcmpi(win, 'Kaiser')
  set(etxt, 'Enable', 'on');
else
  set(etxt, 'Enable', 'off');
end



end

```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = specwin_viewer(varargin)

% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% Check if I exist already
id = findobj('Tag', 'LTPDAspecwin_viewer');
```

```
if ~isempty(id)
  figure(id)
  return
end

%% Some initial setup

Gproperties.Gcol    = [240 240 240]/255;
Gproperties.Gwidth  = 800;
Gproperties.Gheight = 400;
Gproperties.Gborder = 10;
fontsize = getappdata(0, 'ltpda_repo_gui_fontsize');

Gproperties.Screen   = get(0,'screensize');
Gproperties.Gposition = [150 ...
  150 ...
  Gproperties.Gwidth...
  Gproperties.Gheight];

%  Initialize and hide the GUI as it is being constructed.
mainfig = figure('Name', 'LTPDA Spectral Window Viewer',...
  'NumberTitle', 'off',...
  'Visible','off',...
  'Position',Gproperties.Gposition,...
  'Color', Gproperties.Gcol,...
  'ToolBar',       'none',            ...
  'NextPlot',      'new',             ...
  'MenuBar', 'none',...
  'Resize', 'off',...
  'Tag', 'LTPDAspecwin_viewer');

% Set mainfig callbacks
set(mainfig, 'CloseRequestFcn', {'ltpda_specwin_viewer_close', mainfig});

% Set Application data
setappdata(mainfig, 'Gproperties', Gproperties);


%% Window selection dialog

%----------------------------------- Window selection box
buildWindowSelectionMenu(mainfig);

%----------------------------------- Window size

sth = uicontrol(mainfig,'Style','text',...
  'String','Window Size',...
  'BackgroundColor', Gprops.Gcol, ...
  'Position',[10 Gprops.Gheight-70 80 25]);

eth = uicontrol(mainfig,'Style','edit',...
                'String','100',...
                'BackgroundColor', 'w', ...
                'Tag', 'LTPDA_specwin_viewer_WinSize', ...
                'Position',[100 Gprops.Gheight-70 120 25]);

%------------------------------------- PSLL

sth = uicontrol(mainfig,'Style','text',...
  'String','Window PSLL',...
  'BackgroundColor', Gprops.Gcol, ...
  'Position',[10 Gprops.Gheight-100 80 25]);

eth = uicontrol(mainfig,'Style','edit',...
                'String','150',...
                'BackgroundColor', 'w', ...
                'Tag', 'LTPDA_specwin_viewer_PSLL', ...
                'Position',[100 Gprops.Gheight-100 120 25]);

%---------------------------- Build Time-domain Button
pbh = uicontrol(mainfig,'Style','pushbutton',...
  'String','Plot Time-domain',...
  'Callback', {'ltpda_specwin_viewer_build_window', 'Time-domain'}, ...
  'Position',[10 Gprops.Gheight-140 100 25]);

%---------------------------- Build Freq-domain Button
pbh = uicontrol(mainfig,'Style','pushbutton',...
  'String','Plot Freq-domain',...
  'Callback', {'ltpda_specwin_viewer_build_window', 'Freq-domain'}, ...
  'Position',[120 Gprops.Gheight-140 100 25]);

%------------------------- Axes

M = 0.01;
ax   = axes('Position', [0.5+M 0.2 0.4 0.7], ...
            'Parent', mainfig, ...
```

```
            'Tag', 'LTPDA_specwin_viewer_axes', ...
            'Visible', 'on', ...
            'Box', 'on');

setappdata(mainfig, 'axes', ax);

sth = uicontrol(mainfig,'Style','text',...
  'String','None',...
  'BackgroundColor', Gprops.Gcol, ...
  'Tag', 'LTPDA_specwin_viewer_wininfo', ...
  'Position',[10 10 300 Gproperties.Gheight/2 - 20], ...
  'FontSize', 14, 'ForegroundColor', 'b');

% Constructor text field
sth = uicontrol(mainfig,'Style','edit',...
  'String','',...
  'BackgroundColor', 'w', ...
  'Tag', 'LTPDA_specwin_viewer_cstr', ...
  'Position',[10 10 300 25]);


% call callback for window type
ltpda_specwin_viewer_wintype(mainfig);

% draw window
ltpda_specwin_viewer_build_window('Time-domain');




%% Start the GUI

% Make the GUI visible.
set(mainfig,'Visible','on')


%% Sub-functions

  %---- Window selection box
  function buildWindowSelectionMenu(mainfig)

    Gprops = getappdata(mainfig, 'Gproperties');

    % get window list
    wins = specwin.getTypes;

    % text field
    sth = uicontrol(mainfig,'Style','text',...
      'String','Window Type',...
                'BackgroundColor', Gprops.Gcol, ...
      'Position',[10 Gprops.Gheight-40 80 25]);

    % pop-up dialog
    pmh = uicontrol(mainfig,'Style','popupmenu',...
                'String', [{'Kaiser'} wins],...
                'Value', 1, ...
                'BackgroundColor', Gprops.Gcol, ...
                'Tag', 'LTPDA_specwin_viewer_WinType', ...
                'Callback', {'ltpda_specwin_viewer_wintype', mainfig}, ...
                'Position',[100 Gprops.Gheight-40 120 25]);

  end

end
```

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%% path: m/helper %%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%%   path: m/helper   %%%%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/helper/mc">m/helper/mc</a> - A function to properly clear MATLAB
memory for LTPDA.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

Contents

PURPOSE ^

%%%%%%%%%%%%%%%%%%%%% path: m/math %%%%%%%%%%%%%%%%%%%%%

SYNOPSIS ^

This is a script file.

DESCRIPTION ^

```
%%%%%%%%%%%%%%%%%%%   path: m/math   %%%%%%%%%%%%%%%%%%%

   <a href="matlab:help m/math/ltpda_lincom">m/math/ltpda_lincom</a> -  make the linear
combination of the input analysis objects.
```

CROSS-REFERENCE INFORMATION ^

This function calls:

This function is called by:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:
--------------------------------
```

Constructor examples of the XYDATA class Constructor examples of the ZERO class

©LTP Team

< Master index Index for m/ssm_models >

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

---

*Generated on Mon 08-Sep-2008 13:18:35 by **m2html** © 2003*

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 AO.
   <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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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
- % 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:

---

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% SSM_MODEL_DELAY defines a delay block, discrete
%
% VERSION: $Id:$
%
function sys = ssm_model_delay(sys)

  % parameters in subsystem
  sys.paramnames ={'Tau'};
  sys.paramvalues = [0];
  sys.paramsigmas = [0.05];

  %% declaring symbolic parameters
  for i_params=1:length(sys.paramnames)
    cmd = [sys.paramnames{i_params}, '=sym(''', sys.paramnames{i_params},''');'];
    eval(cmd);
  end

  %%%%%%%%%%%%%%%%%%%%%%%%%%%% START BUILDING PARAMETRIC SUBSYTEM %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  Ts = 0.1;
  Tt = 0.3;

  A1 = [...
    0 1 0 0 0 ;...
    0 0 1 0 0 ; ...
    0 0 0 1 0 ; ...
    0 0 0 0 1 ; ...
    0 0 0 0 0];
  B1 = [0; 0; 0; 0; 1];
  C1 = [Tau 1-Tau 0 0 0];
  D1 = 0;
  A = blkdiag(A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1,A1);
  B = blkdiag(B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1,B1);
  C = blkdiag(C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1,C1);
  D = blkdiag(D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1);

  %%%%%%%%%%%%%%%%%%%%%%%%%%%% END BUILDING PARAMETRIC SUBSYTEM %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  sys.name = 'Delay';

  sys.amats    = {A};
  sys.bmats    = {B};
```

```
0043   sys.cmats    = {C};
0044   sys.dmats    = {D};
0045   sys.timestep = 0.1;
0046
0047   sys.ssnames    = {'Delay'};
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*

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 ^

```


function sys = ssm_model_standard_system_noparams(sys)
  % defines a simple 2nd order stable system with a control and a noise input

  sys.name = 'standard test system';

  sys.amats  = {[0 1 ; -0.04 -0.1]};
  sys.cmats  = {[1 0]};
  sys.bmats  = {[0;1] [0 0; 1 0]};
  sys.dmats  = {1 [0 1]};
  sys.timestep = 0;

  sys.ssnames = {'standard test system'};
  sys.ssvarnames = {{'x' 'xdot'}};

  sys.inputnames = {'U' 'N'};
  sys.inputvarnames = {{'Fu'} {'Fn' 'On'}};

  sys.paramnames = {};
  sys.paramvalues = [];
  sys.paramsigmas = [];

  sys.outputnames = {'Y'};
  sys.outputvarnames = {{'y'}};
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 ^

```
% SSM_MODEL_STANDARD_SYSTEM_PARAMS

function sys = ssm_model_standard_system_params(sys)
  sys.name = 'standard test system';

  sys.paramnames  = {'w' 'c' 'c1' 'c2' 'b' 'd'};
  sys.paramvalues = [0.2 0.5 0 0 1 1];
  sys.paramsigmas = [0.2 0.1 1 1 1 1];

  % declaring symbolic parameters
  for i_params=1:length(sys.paramnames)
    cmd = [sys.paramnames{i_params}, '=sym(''', sys.paramnames{i_params},''');'];
    eval(cmd);
  end

  sys.amats    = {[0 1 ; -w*w -w*c]};
  sys.cmats    = {[1+c1 c2]};
  sys.bmats    = {[0;b] [0 0; 1 0]};
  sys.dmats    = {[d] [0 1]};
  sys.timestep = 0;

  sys.ssnames    = {'standard test system'};
  sys.ssvarnames = {{'x' 'xdot'}};

  sys.inputnames    = {'U' 'N'};
  sys.inputvarnames = {{'Fu'} {'Fn' 'On'}};

  sys.outputnames    = {'Y'};
  sys.outputvarnames ={{'y'}};
end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
function varargout = phasetrack(varargin)

ALGONAME = mfilename;
VERSION  = '$Id: phasetrack.m,v 1.1 2008/05/21 09:53:13 anneke Exp $';
CATEGORY = 'Signal Processing';

%% Check if this is a call for parameters, the CVS version string
% or the function category
if nargin == 1 && ischar(varargin{1})
  in = char(varargin{1});
  if strcmp(in, 'Params')
    varargout{1} = getDefaultPL();
    return
  elseif strcmp(in, 'Version')
    varargout{1} = VERSION;
    return
  elseif strcmp(in, 'Category')
    varargout{1} = CATEGORY;
    return
  end
end


invars = {};
for j=1:nargin
  invars = [invars cellstr(inputname(j))];
end



as = [];
pl = [];
for j=1:nargin
  a = varargin{j};
  if isa(a, 'plist')
    pl = a;
  end
  if isa(a, 'ao')
    for k=1:length(a)
      ak = a(k);
      d = ak.data;
      if isa(d, 'tsdata')
        as = [as ak];
```

```
      else
        warning('### works only for time series');
      end
    end
  end
end
na = length(as);
% unpack parameter list
plo = plist();

% Initialise output
bo = [];

for i=1:na

  % get data out
  a = as(i);
  d = a.data;
  dinfo = whos('d');
  if ~isa(d, 'tsdata')
    error('### I only work with time-series at the moment.');
  end
  if ~isreal(d.y)
    error('### I only work with real time-series at the moment.');
  end
  add = 0;
  ydata = a.data.y;
  y = ydata;

  for i = 2:length(ydata)

    diff = ydata(i)-ydata(i-1);
    if diff > pi/2
        add = add-pi;
    elseif diff < -pi/2
        add = add+pi;
    end
    y(i) = ydata(i) + add;
  end
  % Make output analysis object
  nameStr = sprintf('phasetrack(%s)', d.name);

  % create new output data
  data = tsdata(y, d.fs);
  data = set(data, 'name', nameStr);
  data = set(data, 'xunits', d.xunits);
  data = set(data, 'yunits', d.yunits);

  % create new output history
  h = history(ALGONAME, VERSION, pl, a.hist);
  h = set(h, 'invars', invars);

  % make output analysis object
  b = ao(data, h);

  % set name
  % name for this object
  if isempty(invars{j})
    n1 = a.name;
  else
    n1 = invars{j};
  end

  nameStr = sprintf('phasetrack(%s)', n1);
  b = setnh(b, 'name', nameStr);

  % Add to output array
  bo = [bo b];


end
varargout{1} = bo;

%-----------------------------------------------------------------------------
% Get default params
function plo = getDefaultPL()

disp('* creating default plist...');
plo = plist();
disp('* done.');


% END


```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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   - freqeuncy 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 ^

```
% 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   - freqeuncy 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_linedetect(varargin)

warning('!!! This function is deprecated and will be removed from future versions of
LTPDA. Use ao/linedetect now.');
```

```
% Call ao/cohere
bs =  linedetect(varargin{:});

% set output
varargout{1} = bs;


end
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 ^

```
% 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
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = ltpda_nfest(varargin)

  warning('!!! This function is deprecated and will be removed from future versions of LTPDA. Use ao/smoother now.');

  % Call ao/ltfe
```

```
0027   bs =  smoother(varargin{:});
0028
0029   % set output
0030   varargout{1} = bs;
0031
0032 end
0033
0034
```

*Generated on Mon 08-Sep-2008 13:18:47 by **m2html** © 2003*

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 $
```

```
%
% HISTORY:      01-03-2007 M Hewitson
%                  Creation
%
%%%
```

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.');
0024
0025   % Call ao/psd
0026   bs =  rms(varargin{:});
0027
```

```
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 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.

 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 ^

```
% 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 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.
%
% VERSION:      $Id: ltpda_xcorr.m,v 1.7 2008/08/11 10:01:08 hewitson Exp $
%
% HISTORY:      07-02-2007 M Hewitson
%                  Creation
%
%%%
```

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